

Data Warehouse Service

Developer Guide

Issue	01
Date	2025-12-12



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

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

Trademarks and Permissions



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

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

Notice

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

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

Contents

1 Before You Start.....	1
2 Migrating DWS Data.....	5
2.1 Data Migration to DWS.....	5
2.2 Data Type Mapping.....	7
2.2.1 Data Type Mapping Between DWS Foreign Tables and Hive/Spark, ORC, and Parquet Data Sources.....	7
2.2.2 Data Type Mapping for Migration from MySQL to DWS.....	12
2.2.3 Data Type Mapping for Migration from Oracle to DWS.....	14
2.2.4 Data Type Mapping for Migration from Teradata to DWS.....	15
2.3 Importing Data.....	16
2.3.1 Importing Data from OBS in Parallel.....	16
2.3.1.1 Parallel Data Import from OBS.....	17
2.3.1.2 Importing CSV/TXT Data from OBS.....	21
2.3.1.2.1 Creating Access Keys (AK and SK).....	21
2.3.1.2.2 Uploading Data to OBS.....	23
2.3.1.2.3 Creating an OBS Foreign Table.....	25
2.3.1.2.4 Importing Data.....	28
2.3.1.2.5 Handling Import Errors.....	30
2.3.1.3 Importing ORC, CarbonData, Parquet, and JSON Data from OBS.....	33
2.3.1.3.1 Preparing Data on OBS.....	33
2.3.1.3.2 Creating a Foreign Server.....	34
2.3.1.3.3 Creating a Foreign Table.....	38
2.3.1.3.4 Querying Data on OBS Through Foreign Tables.....	41
2.3.1.3.5 Deleting Resources.....	42
2.3.1.3.6 Supported Data Types.....	44
2.3.2 Using GDS to Import Data from a Remote Server.....	47
2.3.2.1 Importing Data In Parallel Using GDS.....	47
2.3.2.2 Preparing Source Data.....	51
2.3.2.3 Installing, Configuring, and Starting GDS.....	52
2.3.2.4 Creating a GDS Foreign Table.....	56
2.3.2.5 Importing Data.....	59
2.3.2.6 Handling Import Errors.....	61
2.3.2.7 Stopping GDS.....	64

2.3.2.8 Example of Importing Data Using GDS.....	64
2.3.3 Importing Data from MRS to a Cluster.....	71
2.3.3.1 Overview.....	71
2.3.3.2 Preparing Data in an MRS Cluster.....	72
2.3.3.3 Manually Creating a Foreign Server.....	75
2.3.3.4 Creating a Foreign Table.....	79
2.3.3.5 Importing Data.....	84
2.3.3.6 Deleting Resources.....	85
2.3.4 GDS-based Cross-Cluster Interconnection.....	87
2.3.5 Using a gsql Meta-Command to Import Data.....	90
2.3.6 Using COPY FROM STDIN to Import Data.....	94
2.3.6.1 Data Import Using COPY FROM STDIN.....	94
2.3.6.2 Introduction to the CopyManager Class.....	94
2.4 Full Database Migration.....	96
2.4.1 Using CDM to Migrate Data to DWS.....	96
2.4.2 Using DSC to Migrate SQL Scripts.....	96
2.5 Metadata Migration.....	97
2.5.1 Using gs_dump and gs_dumpall to Export Metadata.....	97
2.5.1.1 Overview.....	97
2.5.1.2 Exporting a Single Database.....	100
2.5.1.2.1 Exporting a Database.....	100
2.5.1.2.2 Exporting a Schema.....	103
2.5.1.2.3 Exporting a Table.....	106
2.5.1.3 Exporting All Databases.....	109
2.5.1.3.1 Exporting All Databases.....	110
2.5.1.3.2 Exporting Global Objects.....	112
2.5.1.4 Data Export By a User Without Required Permissions.....	114
2.5.2 Using gs_restore to Import Data.....	117
2.6 Exporting Data.....	122
2.6.1 Exporting Data to OBS.....	122
2.6.1.1 Parallel OBS Data Export.....	122
2.6.1.2 Exporting CSV/TXT Data to OBS.....	127
2.6.1.2.1 Planning Data Export.....	127
2.6.1.2.2 Creating an OBS Foreign Table.....	129
2.6.1.2.3 Exporting Data.....	131
2.6.1.2.4 Examples.....	132
2.6.1.3 Exporting ORC and Parquet Data to OBS.....	136
2.6.1.3.1 Planning Data Export.....	136
2.6.1.3.2 Creating a Foreign Server.....	138
2.6.1.3.3 Creating a Foreign Table.....	138
2.6.1.3.4 Exporting Data.....	140
2.6.2 Exporting ORC and Parquet Data to MRS.....	141

2.6.2.1 Overview.....	141
2.6.2.2 Planning Data Export.....	141
2.6.2.3 Creating a Foreign Server.....	143
2.6.2.4 Creating a Foreign Table.....	147
2.6.2.5 Exporting Data.....	149
2.6.3 Using GDS to Export Data to a Remote Server.....	150
2.6.3.1 Exporting Data In Parallel Using GDS.....	150
2.6.3.2 Planning Data Export.....	153
2.6.3.3 Installing, Configuring, and Starting GDS.....	154
2.6.3.4 Creating a GDS Foreign Table.....	157
2.6.3.5 Exporting Data.....	158
2.6.3.6 Stopping GDS.....	159
2.6.3.7 Examples of Exporting Data Using GDS.....	159
2.7 Other Operations.....	164
2.7.1 GDS Pipe FAQs.....	164
2.7.2 Checking for Data Skew.....	166
2.7.3 Analyzing a Table.....	169
3 DWS Development Design Proposal.....	172
3.1 Overview.....	172
3.2 DWS Connection Management Specifications.....	176
3.3 DWS Object Design Specifications.....	178
3.3.1 DATABASE Object Design.....	178
3.3.2 USER Object Design.....	179
3.3.3 Schema Object Design.....	180
3.3.4 TABLESPACE Object Design.....	180
3.3.5 TABLE Object Design (Prioritized).....	180
3.3.6 INDEX Object Design (Prioritized).....	185
3.3.7 VIEW Object Design.....	186
3.4 DWS SQL Statement Development Specifications.....	186
3.4.1 DDL Operations.....	187
3.4.2 INSERT Operation	187
3.4.3 UPDATE and DELETE Operations.....	188
3.4.4 SELECT Operation.....	189
3.5 DWS Foreign Table Function Development Specifications.....	192
3.6 DWS Stored Procedure Development Specifications.....	193
3.7 Detailed Design Rules for DWS Objects.....	195
3.7.1 DWS Database Object Naming Rules.....	195
3.7.2 DWS Database Object Design Rules.....	196
3.7.2.1 DWS Database and Schema Design Rules.....	196
3.7.2.2 DWS Table Design Rules.....	197
3.7.2.3 DWS Column Design Rules.....	199
3.7.2.4 DWS Constraint Design Rules.....	201

3.7.2.5 Design Rules for DWS Views and Associated Tables.....	202
3.7.3 DWS JDBC Configuration Rules.....	202
3.7.4 DWS SQL Writing Rules.....	203
3.7.5 Rules for Using Custom DWS External Functions (pgSQL/Java).....	206
3.7.6 Rules for Using DWS PL/pgSQL.....	207
4 Creating and Managing DWS Database Objects.....	211
4.1 Creating and Managing DWS Databases.....	211
4.2 Creating and Managing DWS Schemas.....	212
4.3 Creating and Managing DWS Tables.....	215
4.4 Selecting a DWS Table Storage Model.....	221
4.5 Creating and Managing DWS Partitioned Tables.....	224
4.6 Creating and Managing DWS Indexes.....	230
4.7 Creating and Using DWS Sequences.....	233
4.8 Creating and Managing DWS Views.....	236
4.9 Creating and Managing DWS Scheduled Tasks.....	237
4.10 Viewing DWS System Catalogs.....	239
5 Syntax Compatibility Differences Among Oracle, Teradata, and MySQL.....	242
6 DWS Database Security Management.....	267
6.1 DWS User and Permissions Management.....	267
6.1.1 DWS Database User Types.....	267
6.1.2 DWS Database User Management.....	268
6.1.3 Creating a Custom Password Policy for DWS.....	270
6.1.4 DWS Database Permissions Management.....	277
6.1.5 Separation of Duties in DWS.....	281
6.2 DWS Sensitive Data Management.....	283
6.2.1 DWS Row-Level Access Control.....	283
6.2.2 DWS Data Masking.....	285
6.2.3 Encrypting and Decrypting DWS Strings.....	289
6.2.4 Using pgcrypto to Encrypt DWS Data.....	292
7 DWS Data Query.....	302
7.1 Single Table Query.....	302
7.2 Multi-Table Join Query.....	303
7.3 Subquery Expressions.....	309
7.4 WITH Expressions.....	312
7.5 UNION Operators.....	319
7.6 SQL on Hudi.....	322
7.6.1 Introduction to Hudi.....	322
7.6.2 Preparations Before Using Hudi.....	323
7.6.3 Hudi User Interfaces.....	324
7.6.4 Creating Hudi Data (Foreign Table).....	326
7.6.5 Synchronizing Hudi Tasks.....	326

7.6.6 Querying a Hudi Foreign Table.....	329
7.6.7 Accessing Hudi Tables on MRS.....	330
8 DWS Sorting Rules.....	332
9 DWS Materialized Views.....	336
9.1 Overview.....	336
9.2 Asynchronous Materialized Views.....	340
9.2.1 Materialized View Query Rewriting.....	340
9.2.1.1 Overview.....	340
9.2.1.2 Text Matching Query Rewriting.....	340
9.2.2 Partitioned Materialized Views.....	342
9.2.2.1 Overview.....	342
9.2.2.2 Creating a Partitioned Materialized View.....	342
9.2.2.3 Refreshing a Partition Materialized View.....	345
9.2.3 Nested Materialized Views.....	347
9.2.3.1 Overview.....	347
9.2.3.2 Creating a Nested Materialized View.....	347
9.2.3.3 Refreshing a Nested Materialized View.....	349
10 DWS User-Defined Functions.....	352
10.1 DWS PL/Java Functions.....	352
10.2 DWS PL/pgSQL Functions.....	364
11 DWS Stored Procedures.....	366
11.1 Overview.....	366
11.2 Data Type Conversion for DWS Stored Procedures.....	366
11.3 DWS Stored Procedure Array and Record.....	367
11.3.1 Arrays.....	368
11.3.2 record.....	374
11.4 Declaration Syntax of DWS Stored Procedures.....	376
11.5 Basic Statements of DWS Stored Procedures.....	378
11.6 Dynamic Statements of DWS Stored Procedures.....	381
11.6.1 Executing Dynamic Query Statements.....	381
11.6.2 Executing Dynamic Non-query Statements.....	384
11.6.3 Dynamically Calling Stored Procedures.....	385
11.6.4 Dynamically Calling Anonymous Blocks.....	387
11.7 Control Statements of DWS Stored Procedure.....	388
11.7.1 RETURN Statements.....	389
11.7.2 Conditional Statements.....	391
11.7.3 Loop Statements.....	393
11.7.4 Branch Statements.....	396
11.7.5 NULL Statements.....	397
11.7.6 Error Trapping Statements.....	397
11.7.7 GOTO Statements.....	399

11.8 Other Statements in DWS Stored Procedures.....	401
11.9 DWS Stored Procedure Cursors.....	402
11.9.1 Overview.....	402
11.9.2 Explicit Cursor.....	402
11.9.3 Implicit Cursor.....	406
11.9.4 Cursor Loop.....	407
11.10 Advanced Packages of DWS Stored Procedures.....	408
11.10.1 DBMS_LOB.....	408
11.10.2 DBMS_RANDOM.....	417
11.10.3 DBMS_OUTPUT.....	418
11.10.4 UTL_RAW.....	420
11.10.5 DBMS_JOB.....	422
11.10.6 DBMS_SQL.....	432
11.11 DWS Stored Procedure Debugging.....	443
12 Using PostGIS Extension.....	446
12.1 PostGIS.....	446
12.2 Using PostGIS.....	447
12.3 PostGIS Support and Constraints.....	447
12.4 OPEN SOURCE SOFTWARE NOTICE (For PostGIS).....	459
13 Using JDBC or ODBC for DWS Secondary Development.....	508
13.1 Prerequisites.....	508
13.2 JDBC-Based Development.....	508
13.2.1 JDBC Development Process.....	508
13.2.2 JDBC Package and Driver Class.....	510
13.2.3 Loading a Driver.....	510
13.2.4 Connecting to a Database.....	510
13.2.5 Executing SQL Statements.....	514
13.2.6 Processing Data in a Result Set.....	517
13.2.7 Common JDBC Development Examples.....	520
13.2.8 Processing RoaringBitmap Result Sets and Importing It to DWS.....	529
13.2.9 JDBC Interfaces.....	532
13.3 ODBC-Based Development.....	546
13.3.1 ODBC Package and Its Dependent Libraries and Header Files.....	547
13.3.2 Configuring a Data Source in the Linux OS.....	548
13.3.3 Configuring a Data Source in the Windows OS.....	556
13.3.4 ODBC Development Example.....	561
13.3.5 ODBC APIs.....	566
14 DWS Hot and Cold Data Management.....	586
15 DWS Resource Monitoring.....	590
15.1 User Resource Monitoring.....	590
15.2 Resource Pool Monitoring.....	592

15.3 Monitoring Memory Resources.....	595
15.4 Instance Resource Monitoring.....	595
15.5 Real-time Top SQL.....	597
15.6 Historical Top SQL.....	602
15.7 Example for Querying for Top SQLs.....	606
16 DWS Performance Tuning.....	610
16.1 Overview.....	610
16.2 Performance Diagnosis.....	612
16.2.1 Cluster Performance Analysis.....	612
16.2.2 Slow SQL Analysis.....	612
16.2.2.1 Querying SQL Statements That Affect Performance Most.....	613
16.2.2.2 Checking Blocked Statements.....	614
16.2.3 SQL Diagnosis.....	615
16.2.4 Table Diagnosis.....	616
16.3 System Optimization.....	617
16.3.1 Tuning Database Parameters.....	617
16.3.2 SMP Parallel Execution.....	625
16.3.3 Configuring LLVM.....	629
16.4 SQL Tuning.....	632
16.4.1 SQL Query Execution Process.....	632
16.4.2 SQL Execution Plan.....	634
16.4.3 Execution Plan Operator.....	646
16.4.4 SQL Tuning Process.....	652
16.4.5 Updating Statistics.....	653
16.4.6 Reviewing and Modifying a Table Definition.....	663
16.4.7 Advanced SQL Tuning.....	664
16.4.7.1 SQL Self-Diagnosis.....	664
16.4.7.2 Optimizing Statement Pushdown.....	668
16.4.7.3 Optimizing Subqueries.....	675
16.4.7.4 Optimizing Statistics.....	683
16.4.7.5 Tuning Operators.....	689
16.4.7.6 Optimizing Data Skew.....	690
16.4.7.7 Proactive Preheating and Tuning of Disk Cache.....	698
16.4.7.8 SQL Statement Rewriting Rules.....	699
16.4.8 Configuring Optimizer Parameters.....	700
16.4.9 Hint-based Tuning.....	702
16.4.9.1 Plan Hint Optimization.....	702
16.4.9.2 Join Order Hints.....	704
16.4.9.3 Join Operation Hints.....	708
16.4.9.4 Rows Hints.....	709
16.4.9.5 Stream Operation Hints.....	710
16.4.9.6 Scan Operation Hints.....	713

16.4.9.7 Sublink Name Hints.....	714
16.4.9.8 Skew Hints.....	715
16.4.9.9 Hint That Disables Subquery Pull-up.....	720
16.4.9.10 Drive Hints.....	721
16.4.9.11 Dictionary Code Hint.....	723
16.4.9.12 Configuration Parameter Hints.....	724
16.4.9.13 Hint Errors, Conflicts, and Other Warnings.....	727
16.4.9.14 Plan Hint Cases.....	729
16.4.10 Routinely Maintaining Tables.....	734
16.4.11 Routinely Recreating an Index.....	736
16.4.12 Automatic Retry upon SQL Statement Execution Errors.....	737
16.4.13 Query Band Load Identification.....	740
16.5 SQL Tuning Examples.....	745
16.5.1 Case: Selecting an Appropriate Distribution Column.....	745
16.5.2 Case: Creating an Appropriate Index.....	746
16.5.3 Case: Adding NOT NULL for JOIN Columns.....	747
16.5.4 Case: Configuring cost_param for Better Query Performance.....	749
16.5.5 Case: Adjusting the Partial Clustering Key.....	752
16.5.6 Case: Adjusting the Table Storage Mode in a Medium Table.....	755
16.5.7 Case: Reconstructing Partition Tables.....	755
16.5.8 Case: Adjusting the GUC Parameter best_agg_plan.....	757
16.5.9 Case: Rewriting SQL Statements and Eliminating Prune Interference.....	758
16.5.10 Case: Rewriting SQL Statements and Deleting in-clause.....	760
16.5.11 Case: Setting Partial Cluster Keys.....	762
16.5.12 Case: Converting from NOT IN to NOT EXISTS.....	764
17 Hybrid Data Warehouse.....	766
17.1 Introduction to Hybrid Data Warehouse.....	766
17.2 Support and Constraints.....	771
17.3 Hybrid Data Warehouse Syntax.....	772
17.3.1 CREATE TABLE.....	772
17.3.2 INSERT.....	781
17.3.3 DELETE.....	783
17.3.4 UPDATE.....	784
17.3.5 UPSERT.....	785
17.3.6 MERGE INTO.....	787
17.3.7 SELECT.....	788
17.3.8 ALTER TABLE.....	790
17.4 Hybrid Data Warehouse Functions.....	792
17.5 Hybrid Data Warehouse Binlog.....	799
17.5.1 Subscribing to Hybrid Data Warehouse Binlog.....	799
17.5.2 Real-Time Binlog Consumption by Flink.....	805
18 DWS System Catalogs and Views.....	811

18.1 Overview of System Catalogs and System Views.....	811
18.2 System Catalogs.....	812
18.2.1 GS_BLOCKLIST_QUERY.....	812
18.2.2 GS_BLOCKLIST_SQL.....	812
18.2.3 GS_OBSCANINFO.....	813
18.2.4 GS_RESPOOL_RESOURCE_HISTORY.....	814
18.2.5 GS_WLM_INSTANCE_HISTORY.....	817
18.2.6 GS_WLM_OPERATOR_INFO.....	818
18.2.7 GS_WLM_SESSION_INFO.....	820
18.2.8 GS_WLM_USER_RESOURCE_HISTORY.....	829
18.2.9 PG_AGGREGATE.....	830
18.2.10 PG_AM.....	831
18.2.11 PG_AMOP.....	833
18.2.12 PG_AMPROC.....	834
18.2.13 PG_ATTRDEF.....	835
18.2.14 PG_ATTRIBUTE.....	836
18.2.15 PG_AUTHID.....	838
18.2.16 PG_AUTH_HISTORY.....	840
18.2.17 PG_AUTH_MEMBERS.....	840
18.2.18 PG_BLOCKLISTS.....	841
18.2.19 PG_CAST.....	842
18.2.20 PG_CLASS.....	843
18.2.21 PG_COLLATION.....	848
18.2.22 PG_CONSTRAINT.....	849
18.2.23 PG_CONVERSION.....	851
18.2.24 PG_DATABASE.....	852
18.2.25 PG_DB_ROLE_SETTING.....	853
18.2.26 PG_DEFAULT_ACL.....	854
18.2.27 PG_DEPEND.....	854
18.2.28 PG_DESCRIPTION.....	856
18.2.29 PG_ENUM.....	857
18.2.30 PG_EXCEPT_RULE.....	857
18.2.31 PG_EXTERNAL_NAMESPACE.....	858
18.2.32 PG_EXTENSION.....	858
18.2.33 PG_EXTENSION_DATA_SOURCE.....	859
18.2.34 PG_FINE_DR_INFO.....	860
18.2.35 PG_FOREIGN_DATA_WRAPPER.....	860
18.2.36 PG_FOREIGN_SERVER.....	861
18.2.37 PG_FOREIGN_TABLE.....	862
18.2.38 PG_INDEX.....	862
18.2.39 PG_INHERITS.....	864
18.2.40 PG_JOB_INFO.....	865

18.2.41 PG_JOBS.....	865
18.2.42 PG_LANGUAGE.....	866
18.2.43 PG_LARGEOBJECT.....	867
18.2.44 PG_LARGEOBJECT_METADATA.....	868
18.2.45 PG_MATVIEW.....	868
18.2.46 PG_MATVIEW_DEPEND.....	869
18.2.47 PG_MATVIEW_PARTITION.....	870
18.2.48 PG_NAMESPACE.....	870
18.2.49 PG_OBJECT.....	871
18.2.50 PG_OBSSCANINFO.....	872
18.2.51 PG_OPCLASS.....	873
18.2.52 PG_OPERATOR.....	874
18.2.53 PG_OPFAMILY.....	874
18.2.54 PG_PARTITION.....	875
18.2.55 PG_PLTEMPLATE.....	878
18.2.56 PG_PROC.....	879
18.2.57 PG_PUBLICATION.....	882
18.2.58 PG_PUBLICATION_NAMESPACE.....	882
18.2.59 PG_PUBLICATION_REL.....	883
18.2.60 PG_RANGE.....	884
18.2.61 PG_REDACTION_COLUMN.....	884
18.2.62 PG_REDACTION_POLICY.....	886
18.2.63 PG_RELFILENODE_SIZE.....	887
18.2.64 PG_RLSPOLICY.....	887
18.2.65 PG_RESOURCE_POOL.....	888
18.2.66 PG_REWRITE.....	889
18.2.67 PG_SECLABEL.....	890
18.2.68 PG_SHDEPEND.....	891
18.2.69 PG_SHDESCRIPTION.....	892
18.2.70 PG_SHSECLABEL.....	892
18.2.71 PG_STATISTIC.....	893
18.2.72 PG_STATISTIC_DATA.....	894
18.2.73 PG_STATISTIC_EXT.....	895
18.2.74 PG_STATISTIC_INFO.....	896
18.2.75 PG_STAT_OBJECT.....	897
18.2.76 PG_SUBSCRIPTION.....	901
18.2.77 PG_SYNONYM.....	902
18.2.78 PG_TABLESPACE.....	903
18.2.79 PG_TRIGGER.....	903
18.2.80 PG_TS_CONFIG.....	904
18.2.81 PG_TS_CONFIG_MAP.....	904
18.2.82 PG_TS_DICT.....	905

18.2.83 PG_TS_PARSER.....	906
18.2.84 PG_TS_TEMPLATE.....	906
18.2.85 PG_TYPE.....	907
18.2.86 PG_USER_MAPPING.....	910
18.2.87 PG_USER_STATUS.....	911
18.2.88 PG_WORKLOAD_ACTION.....	911
18.2.89 PGXC_CLASS.....	912
18.2.90 PGXC_GROUP.....	912
18.2.91 PGXC_NODE.....	914
18.2.92 PLAN_TABLE_DATA.....	916
18.2.93 SNAPSHOT.....	917
18.2.94 TABLES_SNAP_TIMESTAMP.....	917
18.2.95 System Catalogs for Performance View Snapshot.....	918
18.3 System Views.....	919
18.3.1 ALL_ALL_TABLES.....	919
18.3.2 ALL_CONSTRAINTS.....	919
18.3.3 ALL_CONS_COLUMNS.....	920
18.3.4 ALL_COL_COMMENTS.....	920
18.3.5 ALL_DEPENDENCIES.....	920
18.3.6 ALL_IND_COLUMNS.....	921
18.3.7 ALL_IND_EXPRESSIONS.....	922
18.3.8 ALL_INDEXES.....	922
18.3.9 ALL_OBJECTS.....	923
18.3.10 ALL_PROCEDURES.....	923
18.3.11 ALL_SEQUENCES.....	923
18.3.12 ALL_SOURCE.....	924
18.3.13 ALL_SYNONYMS.....	924
18.3.14 ALL_TAB_COLUMNS.....	925
18.3.15 ALL_TAB_COMMENTS.....	926
18.3.16 ALL_TABLES.....	926
18.3.17 ALL_USERS.....	927
18.3.18 ALL_VIEWS.....	927
18.3.19 DBA_DATA_FILES.....	927
18.3.20 DBA_USERS.....	928
18.3.21 DBA_COL_COMMENTS.....	928
18.3.22 DBA_CONSTRAINTS.....	928
18.3.23 DBA_CONS_COLUMNS.....	929
18.3.24 DBA_IND_COLUMNS.....	929
18.3.25 DBA_IND_EXPRESSIONS.....	930
18.3.26 DBA_IND_PARTITIONS.....	930
18.3.27 DBA_INDEXES.....	931
18.3.28 DBA_OBJECTS.....	932

18.3.29 DBA_PART_INDEXES.....	932
18.3.30 DBA_PART_TABLES.....	933
18.3.31 DBA_PROCEDURES.....	934
18.3.32 DBA_SEQUENCES.....	934
18.3.33 DBA_SOURCE.....	934
18.3.34 DBA_SYNONYMS.....	935
18.3.35 DBA_TAB_COLUMNS.....	935
18.3.36 DBA_TAB_COMMENTS.....	936
18.3.37 DBA_TAB_PARTITIONS.....	936
18.3.38 DBA_TABLES.....	938
18.3.39 DBA_TABLESPACES.....	938
18.3.40 DBA_TRIGGERS.....	938
18.3.41 DBA_VIEWS.....	939
18.3.42 DUAL.....	939
18.3.43 GET_ALL_TSC_INFO.....	939
18.3.44 GET_TSC_INFO.....	940
18.3.45 GLOBAL_COLUMN_TABLE_IO_STAT.....	940
18.3.46 GLOBAL_REDO_STAT.....	941
18.3.47 GLOBAL_REL_IOSTAT.....	942
18.3.48 GLOBAL_ROW_TABLE_IO_STAT.....	942
18.3.49 GLOBAL_STAT_DATABASE.....	943
18.3.50 GLOBAL_TABLE_CHANGE_STAT.....	945
18.3.51 GLOBAL_TABLE_STAT.....	946
18.3.52 GLOBAL_WORKLOAD_SQL_COUNT.....	947
18.3.53 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME.....	948
18.3.54 GLOBAL_WORKLOAD_TRANSACTION.....	949
18.3.55 GS_ALL_CONTROL_GROUP_INFO.....	950
18.3.56 GS_BLOCKLIST_QUERY.....	950
18.3.57 GS_BLOCKLIST_SQL.....	951
18.3.58 GS_CLUSTER_RESOURCE_INFO.....	952
18.3.59 GS_COLUMN_TABLE_IO_STAT.....	952
18.3.60 GS_OBS_READ_TRAFFIC.....	953
18.3.61 GS_OBS_WRITE_TRAFFIC.....	953
18.3.62 GS_INSTR_UNIQUE_SQL.....	954
18.3.63 GS_NODE_STAT_RESET_TIME.....	959
18.3.64 GS_OBS_LATENCY.....	959
18.3.65 GS_QUERY_MONITOR.....	960
18.3.66 GS_QUERY_RESOURCE_INFO.....	962
18.3.67 GS_REL_IOSTAT.....	963
18.3.68 GS_RESPOOL_RUNTIME_INFO.....	963
18.3.69 GS_RESPOOL_RESOURCE_INFO.....	964
18.3.70 GS_RESPOOL_MONITOR.....	967

18.3.71 GS_ROW_TABLE_IO_STAT.....	969
18.3.72 GS_SESSION_CPU_STATISTICS.....	970
18.3.73 GS_SESSION_MEMORY_STATISTICS.....	971
18.3.74 GS_SQL_COUNT.....	971
18.3.75 GS_STAT_DB_CU.....	973
18.3.76 GS_STAT_SESSION_CU.....	974
18.3.77 GS_TABLE_CHANGE_STAT.....	974
18.3.78 GS_TABLE_STAT.....	975
18.3.79 GS_TOTAL_NODEGROUP_MEMORY_DETAIL.....	976
18.3.80 GS_USER_MONITOR.....	977
18.3.81 GS_USER_TRANSACTION.....	979
18.3.82 GS_VIEW_DEPENDENCY.....	979
18.3.83 GS_VIEW_DEPENDENCY_PATH.....	980
18.3.84 GS_VIEW_INVALID.....	980
18.3.85 GS_WAIT_EVENTS.....	981
18.3.86 GS_WLM_OPERATOR_INFO.....	982
18.3.87 GS_WLM_OPERATOR_HISTORY.....	984
18.3.88 GS_WLM_OPERATOR_STATISTICS.....	986
18.3.89 GS_WLM_SESSION_INFO.....	989
18.3.90 GS_WLM_SESSION_HISTORY.....	997
18.3.91 GS_WLM_SESSION_STATISTICS.....	1005
18.3.92 GS_WLM_SQL_ALLOW.....	1010
18.3.93 GS_WORKLOAD_SQL_COUNT.....	1010
18.3.94 GS_WORKLOAD_SQL_ELAPSE_TIME.....	1011
18.3.95 GS_WORKLOAD_TRANSACTION.....	1012
18.3.96 MPP_TABLES.....	1012
18.3.97 PG_AVAILABLE_EXTENSION_VERSIONS.....	1013
18.3.98 PG_AVAILABLE_EXTENSIONS.....	1013
18.3.99 PG_BULKLOAD_STATISTICS.....	1014
18.3.100 PG_COMM_CLIENT_INFO.....	1015
18.3.101 PG_COMM_DELAY.....	1016
18.3.102 PG_COMM_STATUS.....	1016
18.3.103 PG_COMM_RECV_STREAM.....	1017
18.3.104 PG_COMM_SEND_STREAM.....	1018
18.3.105 PG_COMM_QUERY_SPEED.....	1020
18.3.106 PG_CONTROL_GROUP_CONFIG.....	1020
18.3.107 PG_CURSORS.....	1020
18.3.108 PG_EXT_STATS.....	1021
18.3.109 PG_GET_INVALID_BACKENDS.....	1023
18.3.110 PG_GET_SENDERS_CATCHUP_TIME.....	1024
18.3.111 PG_GLOBAL_TEMP_ATTACHED_PIDS.....	1024
18.3.112 PG_GROUP.....	1025

18.3.113 PG_INDEXES.....	1025
18.3.114 PG_JOB.....	1026
18.3.115 PG_JOB_PROC.....	1028
18.3.116 PG_JOB_SINGLE.....	1028
18.3.117 PG_LIFECYCLE_DATA_DISTRIBUTE.....	1030
18.3.118 PG_LOCKS.....	1030
18.3.119 PG_LWLOCKS.....	1032
18.3.120 PG_NODE_ENV.....	1033
18.3.121 PG_OS_THREADS.....	1034
18.3.122 PG_POOLER_STATUS.....	1034
18.3.123 PG_PREPARED_STATEMENTS.....	1035
18.3.124 PG_PREPARED_XACTS.....	1036
18.3.125 PG_PUBLICATION_TABLES.....	1036
18.3.126 PG_QUERYBAND_ACTION.....	1037
18.3.127 PG_REPLICATION_SLOTS.....	1037
18.3.128 PG_ROLES.....	1038
18.3.129 PG_RULES.....	1039
18.3.130 PG_RUNNING_XACTS.....	1040
18.3.131 PG_SECLABELS.....	1040
18.3.132 PG_SEQUENCES.....	1041
18.3.133 PG_SESSION_WLMSTAT.....	1042
18.3.134 PG_SESSION_IOSTAT.....	1044
18.3.135 PG_SETTINGS.....	1045
18.3.136 PG_SHADOW.....	1046
18.3.137 PG_SHARED_MEMORY_DETAIL.....	1047
18.3.138 PG_STATS.....	1047
18.3.139 PG_STAT_ACTIVITY.....	1049
18.3.140 PG_STAT_ALL_INDEXES.....	1052
18.3.141 PG_STAT_ALL_TABLES.....	1053
18.3.142 PG_STAT_BAD_BLOCK.....	1055
18.3.143 PG_STAT_BGWRITER.....	1055
18.3.144 PG_STAT_DATABASE.....	1056
18.3.145 PG_STAT_DATABASE_CONFLICTS.....	1057
18.3.146 PG_STATS_EXT_EXPRS.....	1058
18.3.147 PG_STAT_GET_MEM_MBYTES_RESERVED.....	1060
18.3.148 PG_STAT_USER_FUNCTIONS.....	1061
18.3.149 PG_STAT_USER_INDEXES.....	1061
18.3.150 PG_STAT_USER_TABLES.....	1062
18.3.151 PG_STAT_REPLICATION.....	1063
18.3.152 PG_STAT_SYS_INDEXES.....	1064
18.3.153 PG_STAT_SYS_TABLES.....	1065
18.3.154 PG_STAT_XACT_ALL_TABLES.....	1066

18.3.155 PG_STAT_XACT_SYS_TABLES.....	1067
18.3.156 PG_STAT_XACT_USER_FUNCTIONS.....	1067
18.3.157 PG_STAT_XACT_USER_TABLES.....	1068
18.3.158 PG_STATIO_ALL_INDEXES.....	1068
18.3.159 PG_STATIO_ALL_SEQUENCES.....	1069
18.3.160 PG_STATIO_ALL_TABLES.....	1069
18.3.161 PG_STATIO_SYS_INDEXES.....	1070
18.3.162 PG_STATIO_SYS_SEQUENCES.....	1070
18.3.163 PG_STATIO_SYS_TABLES.....	1071
18.3.164 PG_STATIO_USER_INDEXES.....	1071
18.3.165 PG_STATIO_USER_SEQUENCES.....	1072
18.3.166 PG_STATIO_USER_TABLES.....	1072
18.3.167 PG_THREAD_WAIT_STATUS.....	1073
18.3.168 PG_TABLES.....	1085
18.3.169 PG_TDE_INFO.....	1086
18.3.170 PG_TIMEZONE_ABBREVS.....	1087
18.3.171 PG_TIMEZONE_NAMES.....	1087
18.3.172 PG_TOTAL_MEMORY_DETAIL.....	1088
18.3.173 PG_TOTAL_SCHEMA_INFO.....	1090
18.3.174 PG_TOTAL_USER_RESOURCE_INFO.....	1091
18.3.175 PG_USER.....	1093
18.3.176 PG_USER_MAPPINGS.....	1094
18.3.177 PG_VIEWS.....	1095
18.3.178 PG_WLM_STATISTICS.....	1095
18.3.179 PGXC_BULKLOAD_PROGRESS.....	1096
18.3.180 PGXC_BULKLOAD_INFO.....	1097
18.3.181 PGXC_BULKLOAD_STATISTICS.....	1100
18.3.182 PGXC_COLUMN_TABLE_IO_STAT.....	1101
18.3.183 PGXC_COMM_CLIENT_INFO.....	1102
18.3.184 PGXC_COMM_DELAY.....	1103
18.3.185 PGXC_COMM_RECV_STREAM.....	1103
18.3.186 PGXC_COMM_SEND_STREAM.....	1105
18.3.187 PGXC_COMM_STATUS.....	1106
18.3.188 PGXC_COMM_QUERY_SPEED.....	1107
18.3.189 PGXC_DEADLOCK.....	1107
18.3.190 PGXC_DISK_CACHE_STATS.....	1109
18.3.191 PGXC_DISK_CACHE_ALL_STATS.....	1110
18.3.192 PGXC_DISK_CACHE_PATH_INFO.....	1111
18.3.193 PGXC_GET_STAT_ALL_TABLES.....	1112
18.3.194 PGXC_GET_STAT_ALL_PARTITIONS.....	1113
18.3.195 PGXC_GET_TABLE_SKEWNESS.....	1114
18.3.196 PGXC_GLOBAL_TEMP_ATTACHED_PIDS.....	1115

18.3.197 PGXC_GTM_SNAPSHOT_STATUS.....	1115
18.3.198 PGXC_INSTANCE_TIME.....	1116
18.3.199 PGXC_LOCKWAIT_DETAIL.....	1116
18.3.200 PGXC_INSTR_UNIQUE_SQL.....	1118
18.3.201 PGXC_LOCK_CONFLICTS.....	1121
18.3.202 PGXC_LWLOCKS.....	1122
18.3.203 PGXC_MEMORY_DEBUG_INFO.....	1123
18.3.204 PGXC_NODE_ENV.....	1125
18.3.205 PGXC_NODE_STAT_RESET_TIME.....	1126
18.3.206 PGXC_OBS_IO_SCHEDULER_STATS.....	1126
18.3.207 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS.....	1128
18.3.208 PGXC_OS_RUN_INFO.....	1130
18.3.209 PGXC_OS_THREADS.....	1130
18.3.210 PGXC_POOLER_STATUS.....	1130
18.3.211 PGXC_PREPARED_XACTS.....	1131
18.3.212 PGXC_REDO_STAT.....	1131
18.3.213 PGXC_REL_IOSTAT.....	1132
18.3.214 PGXC_REPLICATION_SLOTS.....	1132
18.3.215 PGXC_RESPOOL_RUNTIME_INFO.....	1133
18.3.216 PGXC_RESPOOL_RESOURCE_INFO.....	1134
18.3.217 PGXC_RESPOOL_RESOURCE_HISTORY.....	1137
18.3.218 PGXC_ROW_TABLE_IO_STAT.....	1140
18.3.219 PGXC_RUNNING_XACTS.....	1141
18.3.220 PGXC_SETTINGS.....	1141
18.3.221 PGXC_SESSION_WLMSTAT.....	1143
18.3.222 PGXC_STAT_ACTIVITY.....	1145
18.3.223 PGXC_STAT_BAD_BLOCK.....	1149
18.3.224 PGXC_STAT_BGWRITER.....	1149
18.3.225 PGXC_STAT_DATABASE.....	1150
18.3.226 PGXC_STAT_OBJECT.....	1152
18.3.227 PGXC_STAT_REPLICATION.....	1156
18.3.228 PGXC_STAT_TABLE_DIRTY.....	1157
18.3.229 PGXC_STAT_WAL.....	1160
18.3.230 PGXC_SQL_COUNT.....	1162
18.3.231 PGXC_TABLE_CHANGE_STAT.....	1162
18.3.232 PGXC_TABLE_STAT.....	1163
18.3.233 PGXC_THREAD_WAIT_STATUS.....	1164
18.3.234 PGXC_TOTAL_MEMORY_DETAIL.....	1166
18.3.235 PGXC_TOTAL_SCHEMA_INFO.....	1168
18.3.236 PGXC_TOTAL_SCHEMA_INFO_ANALYZE.....	1168
18.3.237 PGXC_TOTAL_USER_RESOURCE_INFO.....	1169
18.3.238 PGXC_USER_TRANSACTION.....	1172

18.3.239 PGXC_VARIABLE_INFO.....	1173
18.3.240 PGXC_WAIT_DETAIL.....	1174
18.3.241 PGXC_WAIT_EVENTS.....	1176
18.3.242 PGXC_WLM_OPERATOR_HISTORY.....	1177
18.3.243 PGXC_WLM_OPERATOR_INFO.....	1178
18.3.244 PGXC_WLM_OPERATOR_STATISTICS.....	1180
18.3.245 PGXC_WLM_SESSION_INFO.....	1183
18.3.246 PGXC_WLM_SESSION_HISTORY.....	1190
18.3.247 PGXC_WLM_SESSION_STATISTICS.....	1198
18.3.248 PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS.....	1203
18.3.249 PGXC_WLM_USER_RESOURCE_HISTORY.....	1205
18.3.250 PGXC_WLM_WORKLOAD_RECORDS.....	1208
18.3.251 PGXC_WORKLOAD_SQL_COUNT.....	1209
18.3.252 PGXC_WORKLOAD_SQL_ELAPSE_TIME.....	1210
18.3.253 PGXC_WORKLOAD_TRANSACTION.....	1211
18.3.254 PLAN_TABLE.....	1212
18.3.255 PV_FILE_STAT.....	1213
18.3.256 PV_INSTANCE_TIME.....	1213
18.3.257 PV_MATVIEW_DETAIL.....	1214
18.3.258 PV_OS_RUN_INFO.....	1215
18.3.259 PV_REDO_STAT.....	1215
18.3.260 PV_RUNTIME_ATTSTATS.....	1216
18.3.261 PV_RUNTIME_EXPRSTATS.....	1218
18.3.262 PV_RUNTIME_RELSTATS.....	1221
18.3.263 PV_SESSION_MEMORY.....	1222
18.3.264 PV_SESSION_MEMORY_DETAIL.....	1222
18.3.265 PV_SESSION_STAT.....	1224
18.3.266 PV_SESSION_TIME.....	1224
18.3.267 PV_TOTAL_MEMORY_DETAIL.....	1225
18.3.268 REDACTION_COLUMNS.....	1227
18.3.269 REDACTION_POLICIES.....	1228
18.3.270 REMOTE_TABLE_STAT.....	1228
18.3.271 SHOW_TSC_INFO.....	1229
18.3.272 SHOW_ALL_TSC_INFO.....	1230
18.3.273 USER_COL_COMMENTS.....	1231
18.3.274 USER_CONSTRAINTS.....	1231
18.3.275 USER_CONS_COLUMNS.....	1232
18.3.276 USER_INDEXES.....	1232
18.3.277 USER_IND_COLUMNS.....	1232
18.3.278 USER_IND_EXPRESSIONS.....	1233
18.3.279 USER_IND_PARTITIONS.....	1233
18.3.280 USER_JOBS.....	1234

18.3.281 USER_OBJECTS.....	1235
18.3.282 USER_PART_INDEXES.....	1236
18.3.283 USER_PART_TABLES.....	1237
18.3.284 USER_PROCEDURES.....	1237
18.3.285 USER_SEQUENCES.....	1238
18.3.286 USER_SOURCE.....	1238
18.3.287 USER_SYNONYMS.....	1238
18.3.288 USER_TAB_COLUMNS.....	1239
18.3.289 USER_TAB_COMMENTS.....	1240
18.3.290 USER_TAB_PARTITIONS.....	1240
18.3.291 USER_TABLES.....	1241
18.3.292 USER_TRIGGERS.....	1242
18.3.293 USER_VIEWS.....	1242
18.3.294 V\$SESSION.....	1242
18.3.295 V\$SESSION_LONGOPS.....	1243
19 GUC Parameters of the DWS Database.....	1244
19.1 Viewing GUC Parameters.....	1244
19.2 Configuring GUC Parameters.....	1244
19.3 GUC Parameter Usage.....	1246
19.4 Connection and Authentication.....	1247
19.4.1 Connection Settings.....	1247
19.4.2 Security and Authentication (postgresql.conf).....	1248
19.4.3 Communication Library Parameters.....	1250
19.5 Resource Consumption.....	1257
19.5.1 Memory.....	1257
19.5.2 Statement Disk Space Control.....	1266
19.5.3 Kernel Resources.....	1268
19.5.4 Cost-based Vacuum Delay.....	1268
19.5.5 Asynchronous I/O Operations.....	1270
19.5.6 Disk Caching.....	1272
19.6 Parallel Data Import.....	1273
19.7 Write Ahead Logs.....	1275
19.7.1 Settings.....	1275
19.7.2 Checkpoints.....	1278
19.8 HA Replication.....	1279
19.8.1 Primary Server.....	1279
19.9 Query Planning.....	1279
19.9.1 Optimizer Method Configuration.....	1279
19.9.2 Optimizer Cost Constants.....	1296
19.9.3 Genetic Query Optimizer.....	1299
19.9.4 Other Optimizer Options.....	1301
19.10 Error Reporting and Logging.....	1323

19.10.1 Logging Destination.....	1324
19.10.2 Logging Time.....	1324
19.10.3 Logging Content.....	1328
19.11 Runtime Statistics.....	1332
19.11.1 Query and Index Statistics Collector.....	1332
19.11.2 Performance Statistics.....	1337
19.12 Resource Management.....	1338
19.13 Automatic Cleanup.....	1350
19.14 Default Settings of Client Connection.....	1354
19.14.1 Statement Behavior.....	1354
19.14.2 Zone and Formatting.....	1360
19.14.3 Other Default Parameters.....	1364
19.15 Lock Management.....	1365
19.16 Version and Platform Compatibility.....	1369
19.16.1 Compatibility with Earlier Versions.....	1369
19.16.2 Platform and Client Compatibility.....	1373
19.17 Fault Tolerance.....	1417
19.18 Connection Pool Parameters.....	1418
19.19 Cluster Transaction Parameters.....	1420
19.20 Developer Operations.....	1423
19.21 Auditing.....	1444
19.21.1 Audit Switch.....	1444
19.21.2 Operation Audit.....	1445
19.22 Transaction Monitoring.....	1446
19.23 GTM Parameters.....	1447
19.24 Miscellaneous Parameters.....	1447
20 DWS Developer Terms.....	1457
21 SQL Syntax Reference.....	1475
21.1 DWS SQL Overview.....	1475
21.2 DWS SQL Syntax.....	1476
21.3 Differences Between DWS and PostgreSQL.....	1477
21.4 Keyword.....	1480
21.5 Data Types.....	1512
21.5.1 Numeric Types.....	1512
21.5.2 Monetary Types.....	1517
21.5.3 Boolean Type.....	1518
21.5.4 Character Types.....	1518
21.5.5 Binary Data Types.....	1522
21.5.6 Date/Time Types.....	1524
21.5.7 Geometric Types.....	1532
21.5.8 Array.....	1534
21.5.9 Enumeration Type.....	1538

21.5.10 Network Address Types.....	1539
21.5.11 Bit String Types.....	1541
21.5.12 Text Search Types.....	1542
21.5.13 UUID Type.....	1544
21.5.14 JSON Types.....	1545
21.5.15 RoaringBitmap.....	1549
21.5.16 HLL Data Types.....	1549
21.5.17 Object Identifier Types.....	1552
21.5.18 Pseudo-Types.....	1554
21.5.19 Range Types.....	1555
21.5.20 Composite Types.....	1560
21.5.21 Data Types Supported by Column-Store Tables.....	1561
21.5.22 XML.....	1563
21.6 Constant and Macro.....	1564
21.7 Functions and Operators.....	1565
21.7.1 String Processing Functions and Operators.....	1565
21.7.1.1 Overview of String Processing Functions and Operators.....	1565
21.7.1.2 Obtaining the Length of a String.....	1566
21.7.1.3 Truncating a String.....	1570
21.7.1.4 Concatenating Strings.....	1583
21.7.1.5 Replacing and Padding Strings.....	1585
21.7.1.6 Deleting a Specified Character.....	1590
21.7.1.7 Splitting a String.....	1592
21.7.1.8 Converting a String.....	1593
21.7.1.9 Comparing Strings.....	1602
21.7.1.10 Obtaining the Position of a Character or Substring.....	1603
21.7.1.11 Checking If a String Matches a Specific Pattern Using String Matching Functions.....	1605
21.7.2 Binary String Functions and Operators.....	1606
21.7.3 Bit String Functions and Operators.....	1609
21.7.4 Numeric Functions and Operators.....	1611
21.7.4.1 Numeric Operators.....	1611
21.7.4.2 Numeric Functions.....	1615
21.7.5 Date and Time Processing Functions and Operators.....	1624
21.7.5.1 Date and Time Operators.....	1624
21.7.5.2 Date and Time Functions.....	1626
21.7.5.3 EXTRACT.....	1643
21.7.5.4 date_part.....	1648
21.7.5.5 date_format.....	1648
21.7.5.6 time_format.....	1650
21.7.6 SEQUENCE Functions.....	1654
21.7.7 Array Functions and Operators.....	1656
21.7.7.1 Array Operators.....	1656

21.7.7.2 Array Functions.....	1658
21.7.8 Logical Operators.....	1662
21.7.9 Comparison Operators.....	1663
21.7.10 Pattern Matching Operators.....	1663
21.7.11 Aggregate Functions.....	1668
21.7.12 Window Functions.....	1683
21.7.13 Type Conversion Functions.....	1689
21.7.14 JSON/JSONB Functions and Operators.....	1699
21.7.14.1 JSON/JSONB Operators.....	1700
21.7.14.2 JSON/JSONB Functions.....	1704
21.7.15 Security Functions.....	1724
21.7.16 Conditional Expression Functions.....	1734
21.7.17 Range Functions and Operators.....	1738
21.7.17.1 Range Operators.....	1738
21.7.17.2 Range Functions.....	1742
21.7.18 Data Redaction Functions.....	1744
21.7.19 Bitmap Functions and Operators.....	1745
21.7.19.1 Roaring Bitmap Operators.....	1746
21.7.19.2 Bitmap Functions.....	1749
21.7.19.3 Roaring Bitmap Aggregation Functions.....	1755
21.7.19.4 Use Cases.....	1758
21.7.20 UUID Functions.....	1760
21.7.21 Text Search Functions and Operators.....	1762
21.7.21.1 Text Search Operators.....	1762
21.7.21.2 Text Search Functions.....	1764
21.7.21.3 Text Search Debugging Functions.....	1767
21.7.22 HLL Functions and Operators.....	1769
21.7.22.1 HLL Operators.....	1769
21.7.22.2 Hash Functions.....	1771
21.7.22.3 Precision Functions.....	1775
21.7.22.4 Aggregate Functions.....	1776
21.7.22.5 Functional Functions.....	1778
21.7.22.6 Built-in Functions.....	1781
21.7.23 Set Returning Functions.....	1782
21.7.23.1 Series Generating Functions.....	1782
21.7.23.2 Subscript Generating Functions.....	1784
21.7.24 Geometric Functions and Operators.....	1785
21.7.24.1 Geometric Operators.....	1785
21.7.24.2 Geometric Functions.....	1791
21.7.24.3 Geometric Type Conversion Functions.....	1794
21.7.25 Network Address Functions and Operators.....	1798
21.7.25.1 cidr and inet Operators.....	1798

21.7.25.2 Network Address Functions.....	1801
21.7.26 System Information Functions.....	1804
21.7.26.1 Session Information Functions.....	1804
21.7.26.2 Access Privilege Inquiry Functions.....	1810
21.7.26.3 Schema Visibility Inquiry Functions.....	1814
21.7.26.4 System Catalog Information Functions.....	1815
21.7.26.5 System Function Checking Function.....	1822
21.7.26.6 Comment Checking Functions.....	1823
21.7.26.7 Transaction IDs and Snapshots.....	1824
21.7.26.8 Computing Node Group Function.....	1825
21.7.26.9 Lock Information Function.....	1826
21.7.27 System Management Functions.....	1826
21.7.27.1 Configuration Settings Functions.....	1826
21.7.27.2 Universal File Access Functions.....	1827
21.7.27.3 Server Signaling Functions.....	1829
21.7.27.4 Snapshot Synchronization Functions.....	1833
21.7.27.5 Advisory Lock Functions.....	1834
21.7.27.6 Replication Functions.....	1837
21.7.27.7 Resource Management Functions.....	1846
21.7.27.8 Memory Management Functions.....	1860
21.7.27.9 Other Functions.....	1862
21.7.28 Backup and Restoration Control Functions.....	1888
21.7.28.1 Backup Control Functions.....	1888
21.7.28.2 Restoration Control Functions.....	1893
21.7.29 Database Object Functions.....	1897
21.7.29.1 Database Object Size Functions.....	1897
21.7.29.2 Database Object Position Functions.....	1902
21.7.29.3 Partition Management Function.....	1902
21.7.29.4 Collation Version Function.....	1903
21.7.29.5 Hot and Cold Table Functions.....	1904
21.7.30 Residual File Management Functions.....	1906
21.7.30.1 Functions for Obtaining the Residual File List.....	1906
21.7.30.2 Functions for Scanning Residual Files.....	1908
21.7.30.3 Functions for Verifying Residual Files.....	1915
21.7.30.4 Functions for Deleting Residual Files.....	1918
21.7.30.5 Using Residual File Management Functions.....	1920
21.7.31 Statistics Information Functions.....	1923
21.7.32 Trigger Functions.....	1965
21.7.33 XML Functions.....	1966
21.7.33.1 Generating XML Content.....	1966
21.7.33.2 XML Predicates.....	1970
21.7.33.3 Processing XML.....	1972

21.7.33.4 Mapping a Table to XML.....	1974
21.7.34 Call Stack Recording Functions.....	1976
21.7.35 Hudi System Functions.....	1979
21.7.36 Funnel and Retention Functions.....	1986
21.7.37 EXTERNAL SCHEMA System Functions.....	1991
21.7.38 Storage-Compute Decoupling Functions.....	1992
21.7.38.1 Data Sharing Functions.....	1992
21.7.38.2 Function for Obtaining File Information of Tablespaces in the Trash Directory.....	1994
21.7.39 Time Series Functions.....	1994
21.8 Expressions.....	2001
21.8.1 Simple Expressions.....	2001
21.8.2 Conditional Expressions.....	2003
21.8.3 Subquery Expressions.....	2007
21.8.4 Array Expressions.....	2010
21.8.5 Row Expressions.....	2012
21.9 Type Conversion.....	2012
21.9.1 Overview.....	2013
21.9.2 Operators.....	2014
21.9.3 Functions.....	2016
21.9.4 Value Storage.....	2019
21.9.5 UNION, CASE, and Related Constructs.....	2019
21.10 Full Text Search.....	2022
21.10.1 Introduction.....	2022
21.10.1.1 Full-Text Retrieval.....	2023
21.10.1.2 What Is a Document?.....	2024
21.10.1.3 Basic Text Matching.....	2024
21.10.1.4 Configurations.....	2025
21.10.2 Tables and Indexes.....	2026
21.10.2.1 Searching a Table.....	2026
21.10.2.2 Creating an Index.....	2027
21.10.2.3 Constraints on Index Use.....	2029
21.10.3 Controlling Text Search.....	2029
21.10.3.1 Parsing Documents.....	2030
21.10.3.2 Parsing Queries.....	2031
21.10.3.3 Ranking Search Results.....	2032
21.10.3.4 Highlighting Results.....	2034
21.10.4 Additional Features.....	2035
21.10.4.1 Manipulating tsvector.....	2035
21.10.4.2 Manipulating Queries.....	2036
21.10.4.3 Rewriting Queries.....	2037
21.10.4.4 Gathering Document Statistics.....	2038
21.10.5 Parsers	2038

21.10.6 Dictionaries.....	2043
21.10.6.1 Overview.....	2043
21.10.6.2 Stop Words.....	2044
21.10.6.3 Simple Dictionary.....	2044
21.10.6.4 Synonym Dictionary.....	2045
21.10.6.5 Thesaurus Dictionary.....	2047
21.10.6.6 Ispell Dictionary.....	2049
21.10.6.7 Snowball Dictionary.....	2050
21.10.7 Configuration Examples.....	2050
21.10.8 Testing and Debugging Text Search.....	2051
21.10.8.1 Testing a Configuration.....	2052
21.10.8.2 Testing a Parser.....	2053
21.10.8.3 Testing a Dictionary.....	2053
21.10.9 Limitations.....	2054
21.11 System Operation.....	2054
21.12 Transaction Management.....	2055
21.13 DDL Syntax.....	2061
21.13.1 DDL Syntax Overview.....	2061
21.13.2 Databases, Tablespaces, Schemas, and Data Types.....	2068
21.13.2.1 CREATE DATABASE.....	2068
21.13.2.2 ALTER DATABASE.....	2072
21.13.2.3 DROP DATABASE.....	2075
21.13.2.4 CREATE SCHEMA.....	2077
21.13.2.5 ALTER SCHEMA.....	2079
21.13.2.6 DROP SCHEMA.....	2081
21.13.2.7 CREATE EXTERNAL SCHEMA.....	2082
21.13.2.8 ALTER EXTERNAL SCHEMA.....	2086
21.13.2.9 CREATE TYPE.....	2087
21.13.2.10 ALTER TYPE.....	2098
21.13.2.11 DROP TYPE.....	2101
21.13.3 Users and Roles.....	2101
21.13.3.1 CREATE USER.....	2102
21.13.3.2 ALTER USER.....	2110
21.13.3.3 DROP USER.....	2118
21.13.3.4 CREATE ROLE.....	2120
21.13.3.5 ALTER ROLE.....	2130
21.13.3.6 SET ROLE.....	2139
21.13.3.7 DROP ROLE.....	2140
21.13.3.8 CREATE GROUP.....	2141
21.13.3.9 ALTER GROUP.....	2142
21.13.3.10 DROP GROUP.....	2143
21.13.3.11 DROP OWNED.....	2144

21.13.4 Tables and External Data Sources.....	2144
21.13.4.1 CREATE TABLE.....	2144
21.13.4.2 CREATE TABLE PARTITION.....	2188
21.13.4.3 CREATE TABLE AS.....	2225
21.13.4.4 ALTER TABLE.....	2233
21.13.4.5 ALTER TABLE PARTITION.....	2254
21.13.4.6 RENAME TABLE.....	2270
21.13.4.7 DROP TABLE.....	2271
21.13.4.8 CREATE FOREIGN TABLE (SQL on OBS or Hadoop).....	2272
21.13.4.9 CREATE FOREIGN TABLE (for OBS Import and Export).....	2308
21.13.4.10 CREATE FOREIGN TABLE (for GDS Import and Export).....	2328
21.13.4.11 ALTER FOREIGN TABLE (for HDFS or OBS).....	2352
21.13.4.12 ALTER FOREIGN TABLE (GDS Import and Export).....	2354
21.13.4.13 DROP FOREIGN TABLE.....	2356
21.13.4.14 CREATE SERVER.....	2357
21.13.4.15 ALTER SERVER.....	2363
21.13.4.16 DROP SERVER.....	2368
21.13.5 Views and Materialized Views.....	2368
21.13.5.1 CREATE VIEW.....	2369
21.13.5.2 ALTER VIEW.....	2371
21.13.5.3 DROP VIEW.....	2374
21.13.5.4 CREATE MATERIALIZED VIEW.....	2376
21.13.5.5 ALTER MATERIALIZED VIEW.....	2386
21.13.5.6 REFRESH MATERIALIZED VIEW.....	2389
21.13.5.7 DROP MATERIALIZED VIEW.....	2391
21.13.6 Indexes, Sequences, and File Directories.....	2392
21.13.6.1 CREATE INDEX.....	2392
21.13.6.2 ALTER INDEX.....	2406
21.13.6.3 REINDEX.....	2413
21.13.6.4 DROP INDEX.....	2415
21.13.6.5 CREATE SEQUENCE.....	2417
21.13.6.6 ALTER SEQUENCE.....	2422
21.13.6.7 DROP SEQUENCE.....	2425
21.13.6.8 CREATE TEXT SEARCH CONFIGURATION.....	2425
21.13.6.9 ALTER TEXT SEARCH CONFIGURATION.....	2429
21.13.6.10 DROP TEXT SEARCH CONFIGURATION.....	2432
21.13.6.11 CREATE TEXT SEARCH DICTIONARY.....	2433
21.13.6.12 ALTER TEXT SEARCH DICTIONARY.....	2440
21.13.6.13 DROP TEXT SEARCH DICTIONARY.....	2442
21.13.6.14 CLUSTER.....	2443
21.13.7 Stored Procedures, Functions, Triggers, and Cursors.....	2445
21.13.7.1 CREATE FUNCTION.....	2446

21.13.7.2 ALTER FUNCTION.....	2465
21.13.7.3 DROP FUNCTION.....	2469
21.13.7.4 CREATE PROCEDURE.....	2471
21.13.7.5 DROP PROCEDURE.....	2479
21.13.7.6 CREATE TRIGGER.....	2479
21.13.7.7 ALTER TRIGGER.....	2486
21.13.7.8 DROP TRIGGER.....	2487
21.13.7.9 FETCH.....	2488
21.13.7.10 MOVE.....	2492
21.13.7.11 CURSOR.....	2493
21.13.7.12 CLOSE.....	2495
21.13.8 Data Masking and Security.....	2496
21.13.8.1 CREATE ROW LEVEL SECURITY POLICY.....	2496
21.13.8.2 ALTER ROW LEVEL SECURITY POLICY.....	2502
21.13.8.3 DROP ROW LEVEL SECURITY POLICY.....	2504
21.13.8.4 CREATE REDACTION POLICY.....	2505
21.13.8.5 ALTER REDACTION POLICY.....	2509
21.13.8.6 DROP REDACTION POLICY.....	2512
21.13.9 Resource Pools and Filtering Rules.....	2512
21.13.9.1 CREATE RESOURCE POOL.....	2513
21.13.9.2 ALTER RESOURCE POOL.....	2519
21.13.9.3 DROP RESOURCE POOL.....	2521
21.13.9.4 CREATE EXCEPT RULE.....	2522
21.13.9.5 ALTER EXCEPT RULE.....	2526
21.13.9.6 DROP EXCEPT RULE.....	2528
21.13.9.7 CREATE BLOCK RULE.....	2529
21.13.9.8 ALTER BLOCK RULE.....	2537
21.13.9.9 DROP BLOCK RULE.....	2543
21.13.10 Performance and Execution Plans.....	2544
21.13.10.1 CREATE STATISTICS.....	2544
21.13.10.2 DROP STATISTICS.....	2545
21.13.11 Connections, Sessions, and Resource Clearing.....	2546
21.13.11.1 CLEAN CONNECTION.....	2546
21.13.11.2 ALTER SYSTEM KILL SESSION.....	2548
21.13.11.3 ALTER SESSION.....	2549
21.13.11.4 DISCARD.....	2552
21.13.11.5 TRUNCATE.....	2554
21.13.11.6 VACUUM.....	2557
21.13.12 Settings and Comments.....	2566
21.13.12.1 SHOW.....	2566
21.13.12.2 SET.....	2567
21.13.12.3 RESET.....	2569

21.13.12.4 SET CONSTRAINTS.....	2571
21.13.12.5 SET SESSION AUTHORIZATION.....	2572
21.13.12.6 COMMENT.....	2573
21.13.13 Publications and Subscriptions.....	2576
21.13.13.1 CREATE PUBLICATION.....	2576
21.13.13.2 ALTER PUBLICATION.....	2578
21.13.13.3 DROP PUBLICATION.....	2580
21.13.13.4 CREATE SUBSCRIPTION.....	2581
21.13.13.5 ALTER SUBSCRIPTION.....	2584
21.13.13.6 DROP SUBSCRIPTION.....	2585
21.13.14 Other.....	2586
21.13.14.1 CREATE SYNONYM.....	2586
21.13.14.2 ALTER SYNONYM.....	2588
21.13.14.3 DROP SYNONYM.....	2589
21.13.14.4 ALTER LARGE OBJECT.....	2590
21.13.14.5 CREATE BARRIER.....	2591
21.14 DML Syntax.....	2591
21.14.1 DML Syntax Overview.....	2591
21.14.2 CALL.....	2592
21.14.3 COPY.....	2595
21.14.4 DELETE.....	2617
21.14.5 EXPLAIN.....	2620
21.14.6 EXPLAIN PLAN.....	2627
21.14.7 LOCK.....	2629
21.14.8 MERGE INTO.....	2635
21.14.9 INSERT and UPSERT.....	2640
21.14.9.1 INSERT.....	2640
21.14.9.2 UPSERT.....	2652
21.14.10 UPDATE.....	2658
21.14.11 VALUES.....	2667
21.15 DCL Syntax.....	2669
21.15.1 DCL Syntax Overview.....	2669
21.15.2 ALTER DEFAULT PRIVILEGES.....	2669
21.15.3 ANALYZE ANALYSE.....	2672
21.15.4 DEALLOCATE.....	2678
21.15.5 DO.....	2679
21.15.6 EXECUTE.....	2680
21.15.7 EXECUTE DIRECT.....	2681
21.15.8 GRANT.....	2682
21.15.9 PREPARE.....	2691
21.15.10 REASSIGN OWNED.....	2692
21.15.11 REVOKE.....	2693

21.16 DQL Syntax..... 2696

21.16.1 DQL Syntax Overview..... 2696

21.16.2 SELECT..... 2696

21.16.3 SELECT INTO..... 2725

21.17 TCL Syntax..... 2726

21.17.1 TCL Syntax Overview..... 2726

21.17.2 ABORT..... 2727

21.17.3 BEGIN..... 2727

21.17.4 CHECKPOINT..... 2729

21.17.5 COMMIT | END..... 2729

21.17.6 COMMIT PREPARED..... 2730

21.17.7 PREPARE TRANSACTION..... 2731

21.17.8 SAVEPOINT..... 2732

21.17.9 SET TRANSACTION..... 2733

21.17.10 START TRANSACTION..... 2735

21.17.11 ROLLBACK..... 2736

21.17.12 RELEASE SAVEPOINT..... 2737

21.17.13 ROLLBACK PREPARED..... 2738

21.17.14 ROLLBACK TO SAVEPOINT..... 2738

1 Before You Start

Target Readers

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

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

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

Prerequisites

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

- Create a DWS cluster.
- Use the SQL editor or SQL client to connect to the DWS database.

For details about these tasks, see "Creating a DWS Cluster" and "Connecting to a DWS Cluster" in the *Data Warehouse Service (DWS) User Guide*.

Reading Guide

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

- Sections describing the features, functions, and application scenarios of DWS.
- "Getting Started": guides you through creating a data warehouse cluster, creating a database table, uploading data, and testing queries.

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

You can find useful information from the following table for DWS database application development.

Operation	Query Suggestion
Quickly getting started with DWS	Deploy a cluster, connect to the database, and perform some queries by referring to "Creating a Cluster" and "Connecting to a Cluster" in <i>Data Warehouse Service (DWS) User Guide</i>. When you are ready to construct a database, load data to tables and compile the query content to operate the data in the data warehouse. Then, you can return to the <i>Data Warehouse Service Database Developer Guide</i>.
Understand the internal architecture of a DWS data warehouse.	To know more about DWS, go to the DWS homepage.
Learn how to design tables to achieve the excellent performance.	DWS Development Design Proposal introduces the design specifications that should be complied with during the development of database applications. Modeling compliant with these specifications fits the distributed processing architecture of DWS and provides efficient SQL code. To accelerate service execution through optimization, refer to DWS Performance Tuning. Database administrators' experience and judgment play a more significant role in achieving successful performance optimization than instructions and explanations. However, the DWS Performance Tuning section is expected to systematically describe the performance tuning methods for application development personnel and new DWS DBAs.
Loading data	Importing Data describes how to import data to DWS.
Managing users, groups, and database security	DWS Database Security Management covers database security topics.
Monitoring and optimizing system performance	DWS System Catalogs and Views describes the system catalogs where you can query the database status and monitor the query content and process. To learn how to use the DWS console to check the system running status and monitoring metrics, see "Monitoring Clusters" in the <i>Data Warehouse Service (DWS) User Guide</i>.

SQL Syntax Text Conventions

To better understand how to use the syntax, you can refer to the following description of SQL syntax text conventions.

Format	Description
Uppercase characters	Keywords must be in uppercase.
Lowercase characters	Parameters must be in lowercase.
[]	Items in brackets [] are optional.
...	Preceding elements can appear repeatedly.
[x y ...]	One item is selected from two or more options or no item is selected.
{ x y ... }	One item is selected from two or more options.
[x y ...] [...]	You can choose either multiple parameters or no parameters. If you choose multiple parameters, simply separate them with spaces.
[x y ...] [, ...]	You can choose either multiple parameters or no parameters. If you choose multiple parameters, simply separate them with commas (,).
{ x y ... } [...]	You must select at least one parameter. If you select multiple parameters, separate them with spaces.
{ x y ... } [, ...]	You must select at least one parameter. If you select multiple parameters, separate them with commas (,).

Statement

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

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

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

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

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

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE

PROVIDED HEREUNDER IS ON AN "AS-IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

2 Migrating DWS Data

2.1 Data Migration to DWS

You can import data from different sources to DWS. The features of each method are listed in [Table 2-1](#). You can select a method as required. To optimize your data migration and ETL process, we suggest using Cloud Data Migration (CDM) and DataArts Studio in combination. CDM specializes in batch data migration, while DataArts Studio offers a visually intuitive development environment and seamlessly manages the entire ETL process.

NOTE

- OBS, DLI, and MRS are cloud services.
- GDS, DSC, and **gs_restore**, and **gs_dump** are internal tools.

Table 2-1 Import methods

Import Method	Data Source	Description	Advantage
Importing Data from OBS in Parallel	OBS	You can import data in TXT, CSV, ORC, or CarbonData format from OBS to DWS for query, and can remotely read data from OBS. It is recommended for DWS.	This method features high performance and flexible scale-out.
Using GDS to Import Data from a Remote Server	Servers (remote servers)	You can use the GDS tool provided by DWS to import data from the remote server to DWS in parallel. Multiple DN's are used for the import. This method is efficient and applicable to importing a large amount of data to the database.	

Import Method	Data Source	Description	Advantage
Importing Data from MRS to a Cluster	MRS (HDFS)	You can configure a DWS cluster to connect to an MRS cluster. In DWS, read data from the HDFS of MRS.	This method features high performance and flexible scale-out.
GDS-based Cross-Cluster Interconnection	-	GDS is used for data transit to implement data synchronization between multiple clusters.	This method is available for data synchronization between multiple DWS clusters.
Using a gsql Meta-Command to Import Data	Local files	Unlike the SQL COPY statement, the \copy command can read data from or write data into only local files on a gsql client.	This method is easy to operate and suitable for importing a small amount of data to the database.
Using COPY FROM STDIN to Import Data	Other files or databases	When you use Java to develop applications, the CopyManager interface of the JDBC driver is invoked to write data from files or other databases to DWS.	Data is directly written from other databases to DWS. Service data does not need to be stored in files.
Using CDM to Migrate Data to DWS	Databases, NoSQL, file systems, and big data platforms	CDM can migrate various types of data in batches between homogeneous and heterogeneous data sources. CDM migrates data to DWS using the copy method or the GDS parallel import method.	This method supports data import from abundant data sources and is easy-to-operate.
Using DSC to Migrate SQL Scripts	Databases, NoSQL, file systems, and big data platforms	For details, see the documents of the third-party ETL tool. DWS provides the DSC tool to migrate Teradata/Oracle scripts to DWS.	Provides abundant data sources and powerful data conversion capabilities through OBS.

Import Method	Data Source	Description	Advantage
Using gs_dump and gs_dumpall to Export Metadata	• Plain text • Custom • Directory • .tar	gs_dump exports a single database or its objects. gs_dumpall exports all databases or global objects in a cluster. To migrate database information, you can use a tool to import the exported metadata to a target database.	This method is applicable to metadata migration.
Using gs_restore to Import Data	SQL, TMP, and TAR file formats	During database migration, you can export files using gs_dump tool and import them to DWS by using gs_restore. In this way, metadata, such as table definitions and database object definitions, can be imported. The following definitions need to be imported: • All database objects • A single database object • A single schema • A single table	

2.2 Data Type Mapping

2.2.1 Data Type Mapping Between DWS Foreign Tables and Hive/Spark, ORC, and Parquet Data Sources

When data is read from Hive/Spark or third-party components' ORC/Parquet to DWS foreign tables, data incompatibility may occur due to different data types on different platforms. This section describes the data type mapping between DWS foreign tables and these data sources. By configuring correct data types for foreign table columns, you can ensure that data can be accurately identified, processed, and queried in DWS, and that no data is lost.

For details about the foreign table syntax, see "CREATE FOREIGN TABLE (SQL on OBS or Hadoop)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

Data Type Mapping Between DWS Foreign Tables and Hive/Spark Data Sources

Table 2-2 is based on the types supported by DWS foreign tables. Some types supported by Hive/Spark, such as INTERVAL and UNION, are not supported by DWS foreign tables.

Table 2-2 Data type mapping between DWS foreign tables and Hive/Spark data sources

Type	DWS Foreign Table Type	Hive Type	Spark Type
1-byte integer	- TINYINT (not recommended) - SMALLINT (recommended)	TINYINT	BYTE, TINYINT
2-byte integer	SMALLINT	SMALLINT	SHORT, SMALLINT
4-byte integer	INTEGER	INT, INTEGER	INT, INTEGER
8-byte integer	BIGINT	BIGINT	LONG, BIGINT
Single-precision floating point	FLOAT, REAL	FLOAT	FLOA, REAL
Double-precision floating point	FLOAT8, DOUBLE	DOUBLE	DOUBLE
Scientific data	DECIMAL[p (,s)]	DECIMAL	DECIMAL
Date type	DATE	DATE	DATE
Time type	TIMESTAMP WITHOUT TIME ZONE	TIMESTAMP	TIMESTAMP_NTZ
	TIMESTAMP WITH TIME ZONE		TIMESTAMP, TIMESTAMP_LTZ
Boolean	BOOLEAN	BOOLEAN	BOOLEAN
Char	CHAR(n)	CHAR	CHAR(length)
Varchar	VARCHAR(n)	VARCHAR	VARCHAR(length)
	NVARCHAR2		
text	TEXT(CLOB)	STRING	STRING
Binary	BYTEA	BINARY	BINARY
Complex type	JSON	ARRAY	ARRAY
		MAP	MAP

Type	DWS Foreign Table Type	Hive Type	Spark Type
		STRUCT	STRUCT

Data Type Mapping Between DWS Foreign Tables and ORC Data Sources

- ORC types are internal types defined in the ORC format specifications. Generally, you do not need to pay special attention to these types. They can be used as a reference for creating DWS foreign tables if the Hive or Spark types are not clearly defined.
- When DWS foreign tables are written to ORC files, data types are mapped one by one. However, in the **(read)** scenario, in addition to the recommended mapping of types, some other mapping scenarios are supported. [Table 2-3](#) lists these scenarios.

Table 2-3 Data type mapping between DWS foreign tables and ORC data sources

Type	ORC Type	DWS Foreign Table Type (Read)	DWS Foreign Table Type (Write)
Integer	boolean (1 bit)	BOOLEAN	BOOLEAN
	tinyint (8 bit)	• TINYINT (not recommended) • SMALLINT (recommended)	TINYINT (not recommended)
	smallint (16 bit)	SMALLINT	SMALLINT
	int (32 bit)	INTEGER	INTEGER
	bigint (64 bit)	BIGINT	BIGINT
Floating point	float	FLOAT4	FLOAT4
	double	FLOAT8	FLOAT8
String	string	• TEXT (recommended) • VARCHAR (n) • CHAR (n) • DECIMAL[p, (s)]	• TEXT • DECIMAL[p, (s)] • CHAR (n) • VARCHAR (n)
	char	• CHAR (n) (recommended) • VARCHAR (n)	CHAR (n)
	varchar	VARCHAR(n)	VARCHAR (n)

Type	ORC Type	DWS Foreign Table Type (Read)	DWS Foreign Table Type (Write)
Binary	binary	BYTEA	BYTEA
Scientific data	decimal	DECIMAL[p (,s)]	DECIMAL[p (,s)]
Time	timestamp	• TIMESTAMP WITHOUT TIME ZONE • TIMESTAMP WITH TIME ZONE	• TIMESTAMP WITHOUT TIME ZONE • TIMESTAMP WITH TIME ZONE
	timestamp with local time zone	• TIMESTAMP WITHOUT TIME ZONE • TIMESTAMP WITH TIME ZONE	Not supported
	date	• DATE • TIMESTAMP WITHOUT TIME ZONE • TIMESTAMP WITH TIME ZONE	DATE
Complex type	struct	• JSON • JSONB	Not supported
	list	• JSON • JSONB	Not supported
	map	• JSON • JSONB	Not supported
	union	Not supported	Not supported

Data type mapping between DWS foreign tables and PARQUET data sources

PARQUET types are the internal storage types defined in the PARQUET format specifications. PARQUET uses **covered_type** (deprecated) or **logical_type** to further describe type behavior. When DWS data is **written** to PARQUET, only **logical_type** is used. When data is **read** from PARQUET, **covered_type** is compatible.

Table 2-4 Data type mapping between DWS foreign tables and PARQUET data sources

Parquet Type	DWS Foreign Table Type (Read)	DWS Foreign Table Type (Write)
BOOLEAN	BOOLEAN	BOOLEAN
INT32	• TINYINT (not recommended) • SMALLINT • INTEGER • DATE • TIMESTAMP[(p)] • DECIMAL[p (,s)]	• TINYINT (not recommended) • SMALLINT • INTEGER
INT64	• BIGINT • money • TIMESTAMP WITHOUT TIME ZONE] • TIMESTAMP WITH TIME ZONE • DECIMAL[p (,s)]	• BIGINT • DECIMAL[p (,s)] • money • TIMESTAMP WITHOUT TIME ZONE • TIMESTAMP WITH TIME ZONE
INT96	• TIMESTAMP[(p)] [WITHOUT TIME ZONE] • TIMESTAMP WITH TIME ZONE	Not supported
FLOAT	FLOAT4	FLOAT4
DOUBLE	FLOAT8	FLOAT8
BYTE_ARRAY	• CHAR (n) • VARCHAR (n) • TEXT • NVARCHAR2 • BYTEA	• DECIMAL[p (,s)] • CHAR (n) • VARCHAR (n) • TEXT • NVARCHAR2 • BYTEA
FIXED_LEN_BYTE_ARRAY	DECIMAL[p (,s)]	DECIMAL[p (,s)]

2.2.2 Data Type Mapping for Migration from MySQL to DWS

Table 2-5 Data type mapping for migration from MySQL to DWS

MySQL Data Type	DWS Data Type	Mapping Supported or Not
BIT[(M)]	BIT	Yes
TINYINT[(M)]	SMALLINT	Yes
TINYINT[(M)] [UNSIGNED]	SMALLINT	Yes
SMALLINT[(M)]	SMALLINT	Yes
SMALLINT[(M)] [UNSIGNED]	INTEGER	Yes
MEDIUMINT[(M)]	INTEGER	Yes
MEDIUMINT[(M)] [UNSIGNED]	INTEGER	Yes
INT[(M)]	INTEGER	Yes
INT[(M)] [UNSIGNED]	BIGINT	Yes
BIGINT[(M)]	BIGINT	Yes
BIGINT[(M)] [UNSIGNED]	NUMERIC	Yes
DECIMAL[(M[,D])]	SMALLINT / INTEGER / BIGINT / NUMERIC Use a data type that meets the precision and scale requirements.	Yes
FLOAT(p)	FLOAT4	Yes
DOUBLE[(M,D)]	FLOAT8	Yes
DATE	DATE	Yes
DATETIME[(fsp)]	TIMESTAMP	Yes
TIMESTAMP[(fsp)]	TIMESTAMP WITH TIME ZONE	Yes
TIME[(fsp)]	TIME	Yes
YEAR[(4)]	SMALLINT	Yes
CHAR[(M)]	CHAR	Yes
VARCHAR(M)	CHARACTER VARYING() Expand the character set of the source database column by corresponding times in the destination database.	Yes

MySQL Data Type	DWS Data Type	Mapping Supported or Not
BINARY[(M)]	BYTEA	Yes
VARBINARY(M)	BYTEA	Yes
TINYBLOB	BLOB	Yes
TINYTEXT	TEXT	Yes
BLOB	BLOB	Yes
TEXT	TEXT	Yes
MEDIUMBLOB	BLOB	Yes
MEDIUMTEXT	TEXT	Yes
LONGBLOB	BLOB	Yes
LONGTEXT	TEXT	Yes
ENUM	VARCHAR	Yes
SET	VARCHAR	Yes
JSON	JSONB	Yes

 NOTE

- DATE values supported by MySQL range from '1000-01-01' to '9999-12-31'.
DATETIME values supported by MySQL range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
TIMESTAMP values supported by MySQL range from '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.
For details, see the official MySQL documentation.
- TIME values supported by MySQL range from '-838:59:59' to '838:59:59'. For details, see the official MySQL documentation. For DWS, the minimum value of the TIME type is 00:00:00, and the maximum value is 24:00:00. If the TIME value in MySQL is less than 00:00:00 or greater than 24:00:00, the synchronization task will fail.
- YEAR value ranges supported by MySQL are 1901 to 2155 and 0000. For details, see official MySQL documentation. DWS does not have the YEAR type, so the YEAR type of MySQL will be converted to the SMALLINT type.
- If the DATE value in MySQL is '0000', the synchronization task will fail.
- If the data type of a column is INT and the column contains the AUTO_INCREMENT attribute, the data type of the column will be converted to SERIAL during synchronization.

2.2.3 Data Type Mapping for Migration from Oracle to DWS

Table 2-6 Data type mapping for migration from Oracle to DWS

Oracle Data Type	DWS Data Type	Mapping Supported or Not
CHAR	CHAR	Yes
VARCHAR	VARCHAR	Yes
VARCHAR2	VARCHAR	Yes
NCHAR	NCHAR	Yes
NVARCHAR2	NVARCHAR	Yes
NUMBER	NUMBER	Yes
BINARY_FLOAT	REAL	Yes
BINARY_DOUBLE	DOUBLE	Yes
FLOAT	FLOAT/REAL	Yes
DATE	TIMESTAMP	Yes
TIMESTAMP	TIMESTAMP If the precision value (<i>fsp</i>) is specified, the value must be in the range of 0 to 6. Value 0 indicates that there is no decimal part. If the value is not specified, the default precision is 0.	Yes
TIMESTAMP WITH TIME ZONE	TIMESTAMPTZ If the precision value (<i>fsp</i>) is specified, the value must be in the range of 0 to 6. Value 0 indicates that there is no decimal part. If the value is not specified, the default precision is 0.	Yes
TIMESTAMP WITH LOCAL TIME ZONE	TIMESTAMPTZ If the precision value (<i>fsp</i>) is specified, the value must be in the range of 0 to 6. Value 0 indicates that there is no decimal part. If the value is not specified, the default precision is 0.	Yes

Oracle Data Type	DWS Data Type	Mapping Supported or Not
INTERVAL	INTERVAL	Yes
BLOB	BYTEA	Yes
CLOB	CLOB	Yes
NCLOB	TEXT	Yes
BFILE	Not supported	No
LONG	TEXT	Yes
LONG_RAW	BYTEA	Yes
RAW (non-primary key column and non-unique key column)	BYTEA	Yes
RAW (primary key column and unique key column)	VARCHAR	Yes
ROWID	CHARACTER(18)	Yes
UROWID	Not supported	No
XMLTYPE	TEXT	Yes
SDO_GEOMETRY	Not supported	No

2.2.4 Data Type Mapping for Migration from Teradata to DWS

Table 2-7 Data type mapping for migration from Teradata to DWS

Teradata Data Type	DWS Data Type	Mapping Supported or Not
BIGINT	BIGINT	Yes
BYTEINT	SMALLINT	Yes
DECIMAL [(n[,m])]	DECIMAL [(n[,m])]	Yes
DOUBLE PRECISION	DOUBLE PRECISION	Yes
FLOAT	DOUBLE PRECISION	Yes
INT / INTEGER	INTEGER	Yes
NUMBER / NUMERIC	NUMERIC	Yes
NUMBER(n[,m])	NUMERIC (n[,m])	Yes

Teradata Data Type	DWS Data Type	Mapping Supported or Not
REAL	REAL	Yes
SMALLINT	SMALLINT	Yes
CHAR[(n)] / CHARACTER [(n)]	CHAR(n)	Yes
CLOB	CLOB	Yes
LONG VARCHAR	TEXT	Yes
VARCHAR(n) / CHAR VARYING(n) / CHARACTER VARYING(n)	VARCHAR(n)	Yes
DATE	DATE	Yes
TIME [(n)]	TIME [(n)]	Yes
TIME [(n)] WITH TIME ZONE	TIME [(n)] WITH TIME ZONE	Yes
TIMESTAMP [(n)]	TIMESTAMP [(n)]	Yes
TIMESTAMP [(n)] WITH TIME ZONE	TIMESTAMP [(n)] WITH TIME ZONE	Yes
PERIOD(DATE)	daterange	Yes
PERIOD(TIME [(n)])	tsrange [(n)]	Yes
PERIOD(TIME WITH TIME ZONE)	tstzrange	Yes
PERIOD(TIMESTAMP [(n)])	tsrange [(n)]	Yes
PERIOD(TIMESTAMP WITH TIME ZONE)	tstzrange	Yes
BLOB[(n)]	blob	Yes
BYTE[(n)]	bytea	Yes
VARBYTE[(n)]	bytea	Yes

2.3 Importing Data

2.3.1 Importing Data from OBS in Parallel

2.3.1.1 Parallel Data Import from OBS

The object storage service (OBS) is an object-based cloud storage service, featuring data storage of high security, proven reliability, and cost-effectiveness. OBS provides large storage capacity for you to store files of any type.

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

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

Currently, data can be imported using either of the following methods:

- Method 1: You do not need to create a server. Use the default server to create a foreign table. Data in TXT or CSV format is supported. For details, see [Importing CSV/TXT Data from OBS](#).
- Method 2: You need to create a server and use the server to create a foreign table. Data in ORC, CarbonData, text, CSV, Parquet, or JSON format is supported. For details, see [Importing ORC, CarbonData, Parquet, and JSON Data from OBS](#).

NOTICE

- Ensure that no Chinese characters are contained in paths used for importing data to or exporting data from OBS.
- Data cannot be imported to or exported from OBS across regions. Ensure that OBS and the DWS cluster are in the same region.
- To ensure the correctness of data import or export, you need to import or export data from OBS in the same compatibility mode.

If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Overview

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

Advantages:

- The CN only plans and delivers data import tasks, and the DNs execute these tasks. This reduces CN resource usage, enabling the CN to process external requests.
- In this way, the computing capabilities and bandwidths of all the DNs are fully leveraged to import data.
- You can preprocess data before the import.

- Fault tolerance can be configured for data format errors during the data import. You can locate incorrect data based on displayed error information after the data is imported.

Disadvantage:

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

Application Scenario:

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

Related Concepts

- **Source data file:** a TEXT, CSV, ORC, CARBONDATA, PARQUET or JSON file that stores data. to be imported in parallel.
- **OBS:** a cloud storage service used to store unstructured data, such as documents, images, and videos. Data is imported in parallel from the OBS server to DWS.
- **Bucket:** a container storing objects on OBS.
 - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.
 - In OBS, each bucket name must be unique and cannot be changed. A default access control list (ACL) is created with a bucket. Each item in the ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.
 - A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.
- **Object:** a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**.
Generally, objects are managed as files. However, OBS has no file system-related concepts, such as files and folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **Key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.
- **Key:** name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name+Object name*.
- **Metadata:** object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.
 - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.
 - User metadata contains object descriptions specified by users for uploading objects.
- **Data:** object content. OBS does not sense the content and regards it as stateless binary data.

- **Ordinary table:** A database table that stores data imported to data files in parallel. Ordinary tables are classified into row-store tables and column-store tables.
- **Foreign table:** A foreign table is used to identify data in a source data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a source data file.

How Data Is Imported

Figure 2-1 shows how data is imported from OBS. The CN plans and delivers data import tasks. It delivers tasks to each DN by file.

The delivery method is as follows:

In **Figure 2-1**, there are four DNs (DN0 to DN3) and OBS stores six files numbered from t1.data.0 to t1.data.5. The files are delivered as follows:

t1.data.0 -> DN0

t1.data.1 -> DN1

t1.data.2 -> DN2

t1.data.3 -> DN3

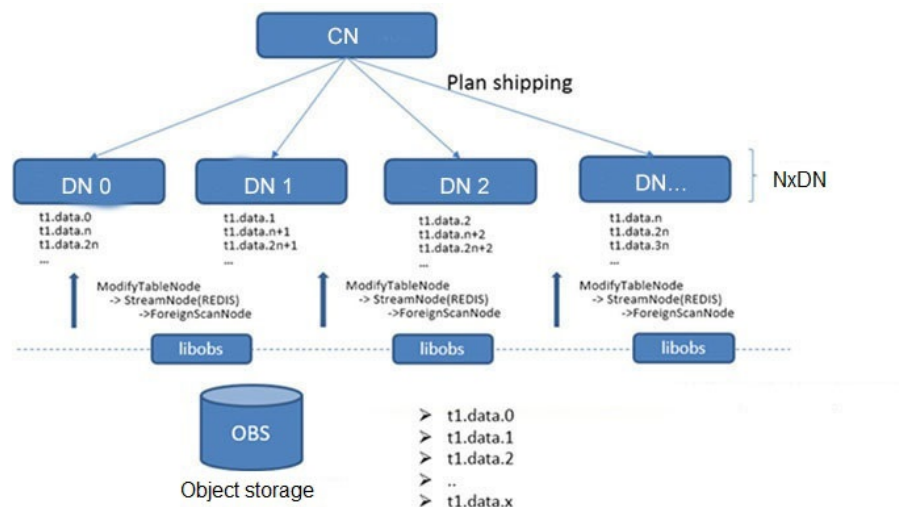
t1.data.4 -> DN0

t1.data.5 -> DN1

Two files are delivered to DN0 and DN1, respectively. One file is delivered to each of the other DNs.

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

Figure 2-1 Parallel data import using OBS foreign tables



Import Flowchart

Figure 2-2 Parallel import procedure

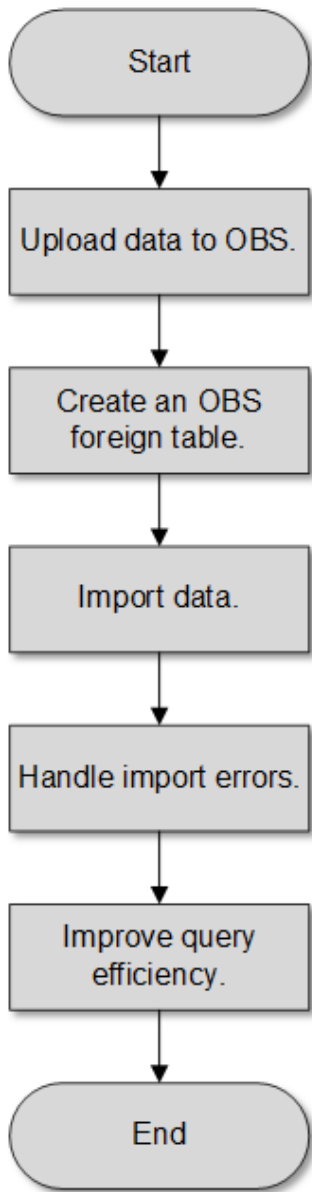


Table 2-8 Procedure description

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

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

2.3.1.2 Importing CSV/TXT Data from OBS

2.3.1.2.1 Creating Access Keys (AK and SK)

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

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

Creating Access Keys (AK and SK)

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

To create an AK/SK pair on the management console, perform the following steps:

Step 1 Log in to the DWS console.

Step 2 Click the username in the upper right corner and choose **My Credentials** from the drop-down list.

Step 3 Select the **Access Keys** tab.

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

NOTE

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

Step 4 Click **Add Access Key**.

Step 5 In the displayed **Add Access Key** dialog box, enter the password and its verification code and click **OK**.

NOTE

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

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

NOTE

- Keep the access keys secure to prevent them from being leaked.
- If you click **Cancel** in the dialog box, the access keys will not be downloaded, and you cannot download them later. In this case, re-create access keys.

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

----End

Precautions

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

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

NOTE

Deleted AK/SK pairs cannot be restored.

2.3.1.2.2 Uploading Data to OBS

Scenarios

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

Preparing Data Files

Prepare source data files to be uploaded to OBS. DWS supports only source data files in CSV, TEXT, ORC, or CarbonData format.

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

NOTE

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

Assume that you have stored the following three CSV files in OBS:

- Data file **product_info.0**

The file contains the following data:

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt
Women,red,M,328,2017-09-04,715,good!
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear
Women,pink,L,584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's
really super nice.
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans
Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite.
```

- Data file **product_info.1**

The file contains the following data:

```
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants
men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress
women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's
really amazing to buy.
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket
women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor.
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat
women,red,L,2004,2017-09-15,826,Very favorite clothes.
```

- Data file **product_info.2**

The file contains the following data:

```
980,"ZKDS-J",2017-09-13,"B","2017 Women's Cotton Clothing","red","M",112,,  
98,"FKQB-I",2017-09-15,"B","2017 new shoes men","red","M",4345,2017-09-18,5473  
50,"DMQY-K",2017-09-21,"A","2017 pants men","red","37",28,2017-09-25,58,"good","good"  
80,"GKLW-I",2017-09-22,"A","2017 Jeans Men","red","39",58,2017-09-25,72,"Very comfortable."  
30,"HWEC-L",2017-09-23,"A","2017 shoes women","red","M",403,2017-09-26,607,"good!"  
40,"IQPD-M",2017-09-24,"B","2017 new pants Women","red","M",35,2017-09-27,52,"very good."  
50,"LPEC-N",2017-09-25,"B","2017 dress Women","red","M",29,2017-09-28,47,"not good at all."  
60,"NQAB-O",2017-09-26,"B","2017 jacket women","red","S",69,2017-09-29,70,"It's beautiful."  
70,"HWNB-P",2017-09-27,"B","2017 jacket women","red","L",30,2017-09-30,55,"I like it so much"  
80,"JKHU-Q",2017-09-29,"C","2017 T-shirt","red","M",90,2017-10-02,82,"very good."
```

Uploading Data to OBS

Step 1 Upload data to OBS.

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

1. Log in to OBS console.

Click **Service List** and choose **Object Storage Service** to open the OBS management console.

2. Create a bucket.

For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Buckets > Creating a Bucket" in the *Object Storage Service User Guide*.

For example, create two buckets named **mybucket** and **mybucket02**.

3. Create a folder.

For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Objects > Creating a Folder" in the *Object Storage Service User Guide*.

For example:

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

4. Upload the files.

For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Objects > Uploading a File" in the *Object Storage Service User Guide*.

For example:

- Upload the following data files to the **input_data** folder in the **mybucket** OBS bucket:

```
product_info.0  
product_info.1
```

- Upload the following data file to the **input_data** folder in the **mybucket02** OBS bucket:

```
product_info.2
```

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

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

The OBS path in the **location** parameter is in the format of **obs://bucket_name/file_path**

For example, the OBS paths are as follows:

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

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

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

For details, see "Console Operation Guide > Permission Control > Configuring a Bucket ACL" in *Object Storage Service User Guide*.

----End

2.3.1.2.3 Creating an OBS Foreign Table

Procedure

Step 1 Based on the path planned in [Uploading Data to OBS](#), determine the value of the **location** parameter used for creating a foreign table.

Step 2 Obtain the keys (AK and SK) to access OBS.

Step 3 Set data format parameters for the foreign table based on the formats of data to be imported. You need to collect the following source data information:

- **format**: format of the source data file in the foreign table. OBS foreign tables support CSV and TEXT formats. The default value is **TEXT**.
- **header**: Whether the data file contains a table header. Only CSV files can have headers.
- **delimiter**: Delimiter specified to separate data fields in a file. If no delimiter is specified, the default one will be used.
- For more parameters that can be identified by foreign tables, see parameters of data format in section "CREATE FOREIGN TABLE (for OBS Import and Export)" in the *Data Warehouse Service (DWS) SQL Syntax Reference*.

Step 4 Plan the error tolerance of parallel import to specify how errors are handled during the import.

- **fill_missing_fields**: When the last column in a row of the source data file is empty, this parameter specifies whether to report an error or set this field in the row to **NULL**.

 NOTE

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

- If this parameter is set to **true** or **on** and the last column of a data row in a source data file is lost, the column will be replaced with **null** and no error message will be generated.
- If this parameter is set to **false** or **off** and the last column of a data row in a source data file is lost, the following error information will be displayed:
missing data for column "tt"

Default value: **false** or **off**

- **ignore_extra_data**: When the number of columns in the source data file is greater than that specified in the foreign table, this parameter specifies whether to report an error or ignore the extra columns.

 NOTE

Value range: true/on, false/off.

- When this parameter is **true** or **on** and the number of data source files exceeds the number of foreign table columns, excessive columns will be ignored.
- If the parameter is set to **false** or **off**, and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:
extra data after last expected column

Default value: **false** or **off**

- **per node reject limit**: This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import fails and an error message will be reported. This parameter is optional.
- **compatible_illegal_chars**: When an illegal character is encountered, this parameter specifies whether to import an error, or convert it and proceed with the import.

 NOTE

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

- When the parameter is **true** or **on**, invalid characters are tolerated and imported to the database after conversion.
- If the parameter is **false** or **off**, and an error occurs when there are invalid characters, the import will be interrupted.

Default value: **false** or **off**

The following describes the rules for converting an invalid character:

- **\0** is converted to a space.
- Other invalid characters are converted to question marks (?).
- If **NULL**, **DELIMITER**, **QUOTE**, or **ESCAPE** is also set to a space or question mark, an error message such as "illegal chars conversion may confuse COPY escape 0x20" is displayed, prompting you to adjust parameter settings to avoid import errors.
- **error_table_name**: This parameter specifies the name of the table that records data format errors. After the parallel import, you can query this table for error details.
- For more parameters, see the fault tolerance parameters in "CREATE FOREIGN TABLE (for OBS Import and Export)" in the *Data Warehouse Service (DWS) SQL Syntax Reference*.

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

----End

Example

Create a foreign table in the DWS database. Parameters are described as follows:

- **Data format parameter access keys (AK and SK)**
 - Set **access_key** to the AK you have obtained.
 - Set **secret_access_key** to the SK you have obtained.

NOTE

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

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

NOTE

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

- **Set fault-tolerant parameters for data import.**
 - To allow all data format errors detected during data import, set the **PER NODE REJECT LIMIT** to **'unlimited'**.
 - To record data format errors detected during data import, set **LOG INTO** to **product_info_err**, which will store the errors in the **product_info_err** table.
 - When **fill_missing_fields** is set to **true** and the last column of a data row in a source data file is missing, it will be replaced with **NULL** and no error message will be displayed.
 - When **ignore_extra_data** is set to **true** and the number of columns in the source data file exceeds the number defined for the foreign table, any extra columns at the end of the row will be ignored and no error message will be displayed.

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

NOTICE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
DROP FOREIGN TABLE product_info_ext;  
  
CREATE FOREIGN TABLE product_info_ext  
(  
    product_price          integer      not null,  
    product_id             char(30)     not null,  
    product_time           date         ,  
    product_level          char(10)     ,  
    product_name           varchar(200) ,  
    product_type1          varchar(20)  ,  
    product_type2          char(10)     ,  
    product_monthly_sales_cnt integer    ,  
    product_comment_time   date         ,  
    product_comment_num    integer      ,  
    product_comment_content varchar(200)  
)  
SERVER gsmpp_server  
OPTIONS(  
  
    LOCATION 'obs://mybucket/input_data/product_info | obs://mybucket02/input_data/product_info',  
    FORMAT 'CSV' ,  
    DELIMITER ',',  
    encoding 'utf8',  
    header 'false',  
    ACCESS_KEY 'access_key_value_to_be_replaced',  
    SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',  
    fill_missing_fields 'true',  
    ignore_extra_data 'true'  
)  
READ ONLY  
LOG INTO product_info_err  
PER NODE REJECT LIMIT 'unlimited';
```

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

```
CREATE FOREIGN TABLE
```

2.3.1.2.4 Importing Data

Background

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

- In most cases, OBS data import performance is limited by concurrent network access rate. Therefore, you are advised to deploy multiple buckets on the OBS server to import data in parallel from buckets, better utilizing DN data transfer.
- Similar to the single table import, ensure that the I/O performance is greater than the maximum network throughput in the concurrent import.
- Set GUC parameters [raise_errors_if_no_files](#), [partition_mem_batch](#), and [partition_max_cache_size](#). When importing data, specify whether to distinguish between the following two cases: no records exist in the data file or the data file does not exist. You also need to specify the number of caches and the size of data buffers.

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

Procedure

Step 1 Create a table in the DWS database to store the data imported from the OBS.

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

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

Step 3 Import data.

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

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

```
INSERT 0 20
```

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

----End

Example

For example, create a table named **product_info**.

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

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

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

2.3.1.2.5 Handling Import Errors

Scenarios

If you encounter an error during data import, use the instructions in this document to resolve it. Note that the error table records only data format errors.

Querying Error Information

Errors that arise during data import are categorized as either data format errors or non-data format errors.

- Data format error

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

```
SELECT * FROM error_table_name;
```

Table 2-9 lists the columns of the **error_table_name** table.

Table 2-9 Columns of the **error_table_name** table

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

- Non-data format error

If a non-data format error occurs during data import, the entire process is halted and the error is not recorded in the error table. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling data import errors

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

Table 2-10 Handling data import errors

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

Error Information	Error Type	Cause	Solution
extra data after last expected column	Format error	The number of columns in the source data file is greater than that in the foreign table.	- Delete the unnecessary columns from the source data file. - When creating a foreign table, set the parameter ignore_extra_data to on. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.
invalid input syntax for type numeric: "a"	Format error	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to numeric .
null value in column "staff_id" violates not-null constraint	Non-format error	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the staff_id column.
duplicate key value violates unique constraint "reg_id_pk"	Non-format error	The unique constraint is violated.	- Delete the duplicate rows from the source data file. - Run the SELECT statement with the DISTINCT keyword to ensure that all imported rows are unique. INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;
value too long for type character varying(16)	Format error	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes.

2.3.1.3 Importing ORC, CarbonData, Parquet, and JSON Data from OBS

2.3.1.3.1 Preparing Data on OBS

Scenarios

Before you use the SQL on OBS feature to query OBS data:

1. You have stored the ORC data on OBS.

For example, the ORC table has been created when you use the Hive or Spark component, and the ORC data has been stored on OBS.

Assume that there are two ORC data files, named **product_info.0** and **product_info.1**, whose original data is stored in the **demo.db/product_info_orc/** directory of the **mybucket** OBS bucket. You can view their original data in [Original Data](#).

2. If your data files are already on OBS, perform steps in [Obtaining the OBS Path of Original Data and Setting Read Permission](#).

NOTE

This section uses the ORC format as an example to describe how to import data. The methods for importing CarbonData, Parquet, and JSON data are similar.

Original Data

Assume that you have stored the two ORC data files on OBS and their original data is as follows:

- Data file **product_info.0**

The file contains the following data:

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt  
Women,red,M,328,2017-09-04,715,good!  
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear  
Women,pink,L,584,2017-09-05,406,very good!  
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.  
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's  
really super nice.  
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans  
Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite.
```

- Data file **product_info.1**

The file contains the following data:

```
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants  
men,black,L,997,2017-09-10,301,The clothes are of good quality.  
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress  
women,black,S,841,2017-09-15,299,Follow the store for a long time.  
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's  
really amazing to buy.  
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket  
women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor.  
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat  
women,red,L,2004,2017-09-15,826,Very favorite clothes.
```

Obtaining the OBS Path of Original Data and Setting Read Permission

Step 1 Log in to OBS console.

Click **Service List** and choose **Object Storage Service** to open the OBS management console.

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

After the source data files are uploaded to an OBS bucket, a globally unique access path is generated. You need to specify the OBS paths of source data files when creating a foreign table.

For details about how to view the OBS path for storing the data source files, see "OBS Console Operation Guide > Managing Objects > Accessing an Object Using Its URL" in the *Object Storage Service User Guide*.

For example, the OBS paths are as follows:

```
https://obs.example.com/mybucket/demo.db/product_info_orc/product_info.0  
https://obs.example.com/mybucket/demo.db/product_info_orc/product_info.1
```

Step 3 Grant the OBS bucket read permission for the user.

The user who executes the SQL on OBS function needs to obtain the read permission on the OBS bucket where the source data file is located. You can configure the ACL for the OBS buckets to grant the read permission to a specific user.

For details, see "Console Operation Guide > Permission Control > Configuring a Bucket ACL" in *Object Storage Service User Guide*.

----End

2.3.1.3.2 Creating a Foreign Server

Create an external server to define the OBS server information for the foreign table. The creation method varies by cluster version.

- For 8.2.0 and later versions, delegate access to OBS bucket data by creating an OBS data source on the OBS console.
- For versions before 8.2.0, you will need to manually create an OBS server. For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

Clusters of Version 8.2.0 or Later

Create an OBS server. For details, see "Managing OBS Data Sources" in the *Data Warehouse Service (DWS) User Guide*.

Clusters of Version Earlier than 8.2.0: (Optional) Creating a User and a Database and Granting Foreign Table Permissions

Common users do not have permissions to create foreign servers and tables. If you want to use a common user to create foreign servers and tables in a customized database, perform the following steps to create a user and a database, and grant the user foreign table permissions.

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

- Step 1** Connect to the default database **gaussdb** as a database administrator through the database client tool provided by DWS.

For example, use the gsql client to connect to the database by running the following command:

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

- Step 2** Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

Switch to the created user.

```
SET ROLE dbuser PASSWORD 'password';
```

Run the following command to create the database demo:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

datname	datdba	encoding	datcollate	datctype	datistemplate	datallowconn	datconnlimit	datlastsysoid	datfrozensid	dattablespace	datcompatibility	datacl
template1	10	0	C	C	t	t	-1	14146	1351			
1663	ORA		{=c/Ruby,Ruby=CTc/Ruby}									
template0	10	0	C	C	t	f	-1	14146	1350			
1663	ORA		{=c/Ruby,Ruby=CTc/Ruby}									
gaussdb	10	0	C	C	f	t	-1	14146	1352			
1663	ORA		{=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,huobinru=C/Ruby}									
mydatabase	17000	0	C	C	f	t	-1	14146	1351			
1663	ORA											

(4 rows)

- Step 3** Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Connect to the new database as a database administrator through the database client tool provided by DWS.

You can use the gsql client to run the following command to switch to an administrator user and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password of the system administrator as prompted.

CAUTION

You must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

```
GRANT ALL ON SCHEMA public TO dbuser;  
GRANT ALL ON FOREIGN DATA WRAPPER dfs_fdw TO dbuser;
```

where *fdw_name* can be **hdfs_fdw** or **dfs_fdw**, and **dbuser** is the name of the user who creates SERVER.

Run the following command to grant the user the permission to use foreign tables:

```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,  
       r.rolcreatorole, r.rolcreatedb, r.rolcanlogin,  
       r.rolconlimit, r.rolvalidbegin, r.rolvaliduntil,  
       ARRAY(SELECT b.rolname  
             FROM pg_catalog.pg_auth_members m  
             JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)  
             WHERE m.member = r.oid) as memberof  
       , r.rolreplication  
       , r.rolauditadmin  
       , r.rolsystemadmin  
       , r.roluseft  
FROM pg_catalog.pg_roles r  
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the UseFT permission.

rolname	rolsuper	rolinherit	rolcreatorole	rolcreatedb	rolcanlogin	rolconlimit	rolvalidbegin	rolvaliduntil	memberof	rolreplication	rolauditadmin	rolsystemadmin	roluseft
dbuser	f	t	f	t	t	-1			{}	f			f
lily	f	t	f	t	t	-1			{}	f			f
Ruby	t	t	t	t	t	-1			{}	t			t

----End

Clusters of Version Earlier than 8.2.0: Creating an External Server

Step 1 Use the user who is about to create a foreign server to connect to the corresponding database.

In this example, use common user **dbuser** created in [Clusters of Version Earlier than 8.2.0: \(Optional\) Creating a User and a Database and Granting Foreign Table Permissions](#) to connect to **mydatabase** created by the user. You need to connect to the database through the database client tool provided by DWS.

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:
`\c mydatabase dbuser,`
Enter the password as prompted.
- If you have not logged in to the gsql client or have exited the gsql client by running the `\q` command, run the following command to reconnect to it:
`gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r`
Enter the password as prompted.

Step 2 Create a foreign server.

Run the commands below to create a foreign server named **obs_server**.

NOTE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER dfs_fdw
OPTIONS (
  address 'obs.example.com',
  ACCESS_KEY 'access_key_value_to_be_replaced',
  SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
  encrypt 'on',
  type 'obs'
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*
You can customize a name.
In this example, the name is set to **obs_server**.
- **FOREIGN DATA WRAPPER**
fdw_name can be **hdfs_fdw** or **dfs_fdw**, which already exists in the database.
- **OPTIONS parameters**
 - **address**
Specifies the endpoint of the OBS service.
Obtain the address as follows:
 - i. Obtain the OBS path by performing [Step 2 in Preparing Data on OBS](#).
 - ii. The OBS path on OBS is **obs.example.com**, which is the endpoint of OBS.
 - (Mandatory) **Access keys (AK and SK)**
DWS needs to use access keys (AK) and secret keys (SK) to access OBS. Therefore, you must obtain the keys first.
 - (Mandatory) **access_key**: specifies users' AK information.
 - (Mandatory) **secret_access_key**: specifies users' SK information.For details about how to obtain the access keys, see [Creating Access Keys \(AK and SK\)](#).

– **type**

Its value is **obs**, which indicates that **dfs_fdw** connects to OBS.

Step 3 View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE srvname='obs_server';
```

The server is successfully created if the returned result is as follows:

```

srvname | srvowner | srvfdw | srvtype | srvversion | srvacl
-----+-----+-----+-----+-----+-----
                                srvoptions
-----+-----+-----+-----+-----+-----
{address=xxx.xxx.x.xxx,access_key=xxxxxxxxxxxxxxxxxxxxx,type=obs,secret_access_key=xxxxxxxxxxxxxxxxxxxxx
xxxxxxxxxxx}
(1 row)
```

----End

2.3.1.3.3 Creating a Foreign Table

After performing steps in [Creating a Foreign Server](#), create an OBS foreign table in the DWS database to access the data stored in OBS. An OBS foreign table is read-only. It can only be queried using **SELECT**. The operations vary according to the cluster version.

- For 8.2.0 and later versions, you can refer to the operations described in "Managing OBS Data Sources" in *Data Warehouse Service (DWS) User Guide*.
- For versions earlier than 8.2.0, refer to the operations outlined in the following section.

CAUTION

If error message "permission denied for foreign server xxx" is displayed during foreign table creation, the current user does not have permissions on the foreign server. To grant the user the permissions on the foreign server, run the following command (replace *obs_server* with the foreign server name and **u1** with the current user name):

```
GRANT usage ON foreign server obs_server TO u1;
```

Creating a Foreign Table

The syntax for creating a foreign table is as follows:

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
    column_constraint [...] } ] |
  table_constraint [, ...] [, ...] } )
SERVER dfs_server
OPTIONS ( { option_name 'value' } [, ...] )
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```


For example, when creating a foreign table **product_info_ext_obs**, configure the parameters in the syntax as follows.

- **table_name**

Specifies the name of the foreign table to be created.

- **Table column definitions**

- **column_name**: specifies the name of a column in the foreign table.
- **type_name**: specifies the data type of the column.

Multiple columns are separate by commas (,).

The number of fields and field types in the foreign table must be the same as those in the data stored on OBS.

- **SERVER dfs_server**

This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign server connects to OBS to read data by setting its foreign server.

Enter the name of the foreign server created by following steps in [Creating a Foreign Server](#).

- **OPTIONS parameters**

These are parameters associated with the foreign table. The key parameters are as follows:

- **format**: indicates the file format on OBS. The ORC, CarbonData, and Parquet formats are supported.
- **foldername**: This parameter is mandatory. It indicates the OBS path of the data source file. You only need to enter */Bucket name/Folder directory level/*.

You can use [Step 2 in Preparing Data on OBS](#) to obtain the complete OBS path of the data source file. The path is the endpoint of OBS.

- **totalrows**: This parameter is optional. It does not indicate the total rows of the imported data. Because OBS may store many files, it is slow to analyze data. This parameter allows you to set an estimated value so that the optimizer can estimate the table size according to the value. Generally, query efficiency is relatively high when the estimated value is almost the same as the actual value.
- **encoding**: encoding of data source files in foreign tables. The default value is **utf8**. This parameter is mandatory for OBS foreign tables.

- **DISTRIBUTE BY:**

This clause is mandatory. The **ROUNDROBIN** and **REPLICATION** distribution modes are supported. By default, the **ROUNDROBIN** distribution mode is used.

ROUNDROBIN indicates that when a foreign table reads data from the data source, each node in the DWS cluster randomly reads some data and integrates the random data to a complete data set.

REPLICATION indicates that when a foreign table reads data from the data source, the DWS cluster selects a node to read all data. This is because each data node has complete table data.

- **Other parameters in the syntax**

Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

Based on the preceding settings, the command for creating the foreign table is as follows:

Create an OBS foreign table that does not contain partition columns. The foreign server associated with the table is **obs_server**, the file format on OBS corresponding to the table is ORC, and the data storage path on OBS is / **mybucket/demo.db/product_info_orc/**.

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price          integer      not null,
    product_id             char(30)    not null,
    product_time           date        ,
    product_level          char(10)    ,
    product_name           varchar(200) ,
    product_type1          varchar(20) ,
    product_type2          char(10)    ,
    product_monthly_sales_cnt integer    ,
    product_comment_time   date        ,
    product_comment_num    integer     ,
    product_comment_content varchar(200)
) SERVER obs_server
OPTIONS (
    format 'orc',
    foldername '/mybucket/demo.db/product_info_orc/',
    encoding 'utf8',
    totalrows '10'
)
DISTRIBUTE BY ROUNDROBIN;
```

Create an OBS foreign table that contains partition columns. The **product_info_ext_obs** foreign table uses the **product_manufacturer** column as the partition key. The following partition directories exist in **obs/mybucket/demo.db/product_info_orc/**:

Partition directory 1: product_manufacturer=10001

Partition directory 2: product_manufacturer=10010

Partition directory 3: product_manufacturer=10086

```
...
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
CREATE FOREIGN TABLE product_info_ext_obs
(
    product_price          integer      not null,
    product_id             char(30)    not null,
    product_time           date        ,
    product_level          char(10)    ,
    product_name           varchar(200) ,
    product_type1          varchar(20) ,
    product_type2          char(10)    ,
    product_monthly_sales_cnt integer    ,
    product_comment_time   date        ,
    product_comment_num    integer     ,
    product_comment_content varchar(200) ,
    product_manufacturer   integer
) SERVER obs_server
OPTIONS (
    format 'orc',
    foldername '/mybucket/demo.db/product_info_orc/',
    encoding 'utf8',
    totalrows '10'
)
```

```
)  
DISTRIBUTE BY ROUNDROBIN  
PARTITION BY (product_manufacturer) AUTOMAPPED;
```

2.3.1.3.4 Querying Data on OBS Through Foreign Tables

Viewing Data on OBS by Directly Querying the Foreign Table

If the data amount is small, you can directly run **SELECT** to query the foreign table and view the data on OBS.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM product_info_ext_obs;
```

If the query result is the same as the data in [Original Data](#), the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

Step 1 Create a table in DWS to store imported data.

The target table structure must be the same as the structure of the foreign table created in [Creating a Foreign Table](#). That is, both tables must have the same number of columns and column types.

For example, create a table named *product_info*. The table example is as follows:

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

Step 2 Run the **INSERT INTO.. SELECT ..** command to import data from the foreign table to the target table.

Example:

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

If information similar to the following is displayed, the data has been imported.
INSERT 0 10

Step 3 Run the following **SELECT** command to view data imported from OBS to DWS:

```
SELECT * FROM product_info;
```

If the query result is the same as the data in [Original Data](#), the import is successful. The following information is displayed at the end of the query result:

```
(10 rows)
```

```
----End
```

2.3.1.3.5 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation. The procedure is as follows:

1. [Deleting the Foreign Table and Target Table](#)
2. [Deleting the Created Foreign Server](#)
3. [Deleting the Database and the User to Which the Database Belongs](#)

If you have performed steps in [Clusters of Version Earlier than 8.2.0: \(Optional\) Creating a User and a Database and Granting Foreign Table Permissions](#), delete the database and the user to which the database belongs.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If operations in [Querying Data After Importing It](#) have been performed, run the following command to delete the target table:

```
DROP TABLE product_info;
```

If the following information is displayed, the table has been deleted.

```
DROP TABLE
```

Step 2 Run the following statement to delete the foreign table:

```
DROP FOREIGN TABLE product_info_ext_obs;
```

If the following information is displayed, the table has been deleted.

```
DROP FOREIGN TABLE
```

```
----End
```

Deleting the Created Foreign Server

Step 1 Use the user who created the foreign server to connect to the database where the foreign server is located.

In this example, common user **dbuser** is used to create the foreign server in **mydatabase**. You need to connect to the database through the database client tool provided by DWS. You can use the gsql client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:

```
\c mydatabase dbuser;
```

Enter the password as prompted.

- If you have logged in to the gsql client, you can run the `\q` command to exit gsql, and run the following command to reconnect to it:

```
gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
```


Enter the password as prompted.

Step 2 Delete the created foreign server.

Run the following command to delete it:

```
DROP SERVER obs_server;
```

The database is deleted if the following information is displayed:

```
DROP SERVER
```

View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE srvname='obs_server';
```

The server is successfully deleted if the returned result is as follows:

```
srvname | srvowner | srvfdw | srvtype | srvversion | srvacl | srvoptions  
-----+-----+-----+-----+-----+-----+-----  
(0 rows)
```

----End

Deleting the Database and the User to Which the Database Belongs

If you have performed steps in [Clusters of Version Earlier than 8.2.0: \(Optional\) Creating a User and a Database and Granting Foreign Table Permissions](#), perform the following steps to delete the database and the user to which the database belongs.

Step 1 Delete the customized database.

Connect to the default database gaussdb through the database client tool provided by DWS.

If you have logged in to the database using the gsql client, run the following command to switch the database and user:

Switch to the default database.

```
\c gaussdb
```

Enter your password as prompted.

Run the following command to delete the customized database:

```
DROP DATABASE mydatabase;
```

The database is deleted if the following information is displayed:

```
DROP DATABASE
```

Step 2 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by DWS.

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c gaussdb dbadmin
```

Run the following command to reclaim the permission for creating foreign servers:
REVOKE ALL ON FOREIGN DATA WRAPPER dfs_fdw FROM dbuser;

The name of **FOREIGN DATA WRAPPER** must be **dfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to delete the user:

```
DROP USER dbuser;
```

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

2.3.1.3.6 Supported Data Types

In the big data field, the mainstream file formats are ORC and Parquet. You can use Hive to export data to an ORC or Parquet file and use DWS to query and analyze the data in the ORC or Parquet file through a read-only foreign table. Therefore, you need to map the data types supported by the ORC or Parquet file format with the data types supported by DWS. For details, see [Table 2-11](#). Similarly, DWS exports data through a write-only foreign table, and stores the data in the ORC or Parquet format. Using Hive to read the ORC or Parquet file also requires matched data types. [Table 2-12](#) shows the matching relationship.

Table 2-11 Mapping between ORC or Parquet read-only foreign tables and Hive data types

Type	DWS Foreign Table Type	Hive Table Type
1-byte integer	TINYINT (not recommended)	TINYINT
	SMALLINT (recommended)	TINYINT
2-byte integer	SMALLINT	SMALLINT
4-byte integer	INTEGER	INT
8-byte integer	BIGINT	BIGINT
Single-precision floating point number	FLOAT4 (REAL)	FLOAT
Double-precision floating point number	FLOAT8(DOUBLE PRECISION)	DOUBLE
Scientific data type	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	DECIMAL (The maximum precision can reach up to 38.) (HIVE 0.11)
Date type	DATE	DATE

Type	DWS Foreign Table Type	Hive Table Type
Time type	TIMESTAMP	TIMESTAMP
Boolean type	BOOLEAN	BOOLEAN
CHAR type	CHAR(n)	CHAR (n)
VARCHAR type	VARCHAR(n)	VARCHAR (n)
String (large text object)	TEXT(CLOB)	STRING
Binary type	BYTEA	BINARY

 NOTE

Binary types are available only for clusters of version 9.1.0.100 or later.

Table 2-12 Mapping between ORC or Parquet write-only foreign tables and Hive data types

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
1-byte integer	TINYINT	TINYINT (not recommended)	SMALLINT
		SMALLINT (recommended)	SMALLINT
2-byte integer	SMALLINT	SMALLINT	SMALLINT
4-byte integer	INTEGER, BINARY_INTEGER	INTEGER	INT
8-byte integer	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4, REAL	FLOAT4, REAL	FLOAT
Double-precision floating point number	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
Scientific data type	DECIMAL, NUMERIC	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	<i>precision</i> ≤ 38: DECIMAL; <i>precision</i> > 38: STRING
Date type	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
Time type	TIME [(p)] [WITHOUT TIME ZONE], TIME [(p)] [WITH TIME ZONE]	TEXT	STRING
	TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
	INTERVAL DAY (l) TO SECOND (p), INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
Boolean type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n), CHARACTER(n), NCHAR(n)	CHAR(n), CHARACTER(n), NCHAR(n)	<i>n</i> ≤ 255: CHAR(n); <i>n</i> > 255: STRING
VARCHAR type	VARCHAR(n), CHARACTER VARYING(n), VARCHAR2(n)	VARCHAR(n)	<i>n</i> ≤ 65535: VARCHAR(n); <i>n</i> > 65535: STRING
	NVARCHAR2(n)	TEXT	STRING
String (large text object)	TEXT, CLOB	TEXT, CLOB	STRING
Binary type	BYTEA	BYTEA	BINARY

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
Monetary type	MONEY	NUMERIC	BIGINT

NOTICE

1. The DWS foreign table supports the NULL definition, and the Hive data table supports and uses the corresponding NULL definition.
2. The TINYINT value range in a Hive data table is [-128, 127] while in DWS it is [0, 255]. To avoid discrepancies between the read and actual values, use the SMALLINT type when creating a DWS read-only foreign table for TINYINT in the Hive table. Similarly, when exporting data of the TINYINT type from DWS, you are advised to use the SMALLINT type for write-only foreign tables and Hive tables.
3. The time zone definition is not supported by the date and time types of the DWS foreign table, or by the Hive table.
4. The DATE type in Hive contains only date. The DATE type in DWS contains date and time.
5. In DWS, ORC files can be compressed in ZLIB, SNAPPY, LZ4, or NONE mode. The FLOAT4 format itself is not accurate, and the sum operation results in different effect in various environments. You are advised to use the DECIMAL type in the high-precision scenarios.
6. In Teradata-compatible mode, foreign tables do not support the DATE type.
7. Binary types are available only for clusters of version 9.1.0.100 or later.

2.3.2 Using GDS to Import Data from a Remote Server

2.3.2.1 Importing Data In Parallel Using GDS

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

In the current GDS version, you can import data to databases from pipe files.

- When the local disk space of the GDS user is insufficient, HDFS data can be directly written to the pipe file without occupying extra disk space.
- To clean data before importing, you can compile a program as needed and write the data to be processed into a pipe file.

NOTE

- The current version does not support data import through GDS in SSL mode. Do not use GDS in SSL mode.
- All pipe files mentioned in this section refer to named pipes on Linux.
- To ensure the correctness of data import or export using GDS, you need to import or export data in the same compatibility mode.
If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Overview

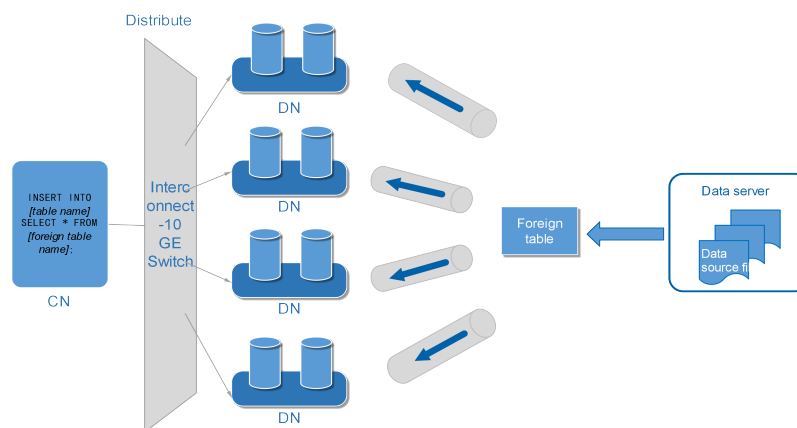
You can import data in parallel from the common file system (excluding HDFS) of a server to DWS.

Data files to be imported are specified based on the import policy and data formats set in a foreign table. Data is imported in parallel through multiple DNs from source data files to the database, which improves the overall data import performance. [Figure 2-3](#) shows an example.

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

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

Figure 2-3 Importing data in parallel



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

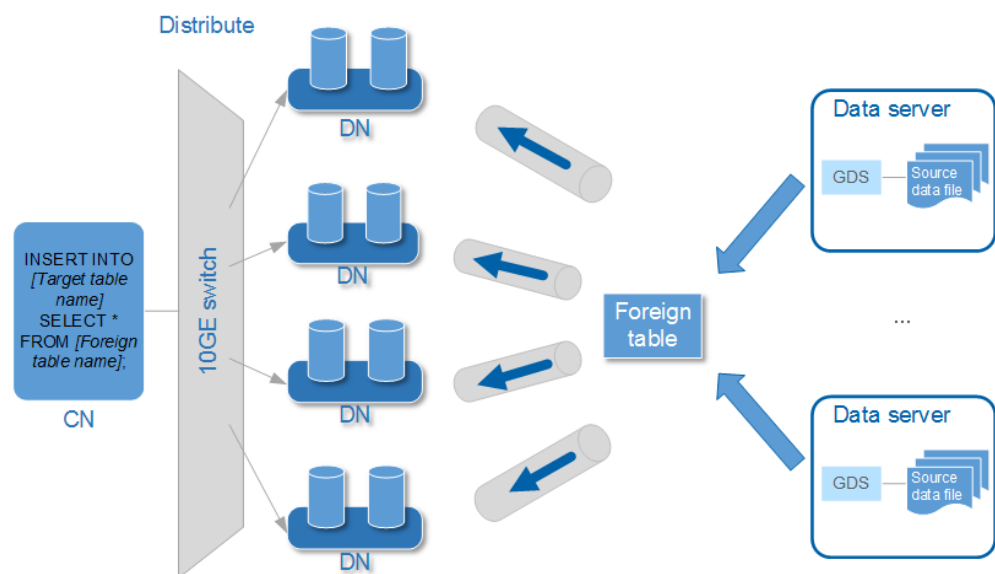
- **CN**: coordinator of DWS. After receiving import SQL requests from an application or client, the CN plans import tasks and delivers the tasks to DNs.
- **DN (Datanode)**: data node of DWS. After receiving the import tasks delivered by the CN, DNs import data from the source data file to the target table in the database through a foreign table.
- **Source data file**: a file that stores data to be imported.
- **Data server**: a server that stores source data files. For security purposes, it is recommended that the data server and DWS be on the same intranet.

- **Foreign table:** a table that stores information, such as the source location, format, destination location, encoding format, and data delimiter of a source data file. It is used to associate source data files with the target table.
- **Target table:** a table in the database. It can be a row-store table or column-store table. Data in the source data files will be imported to this table.

Parallel Import Using GDS

- If a large volume of data is stored on multiple servers, deploy, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel, as shown in [Figure 2-4](#).

Figure 2-4 Parallel import from multiple data servers



NOTICE

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

- If data is stored on data servers, and both DWS and the data servers have available I/O resources, you can use GDS for multi-thread concurrent import. GDS determines the number of threads based on the number of concurrent import transactions. Even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.

Multi-thread concurrent import enables you to:

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

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

Import Process

Figure 2-5 Concurrent import procedure of GDS

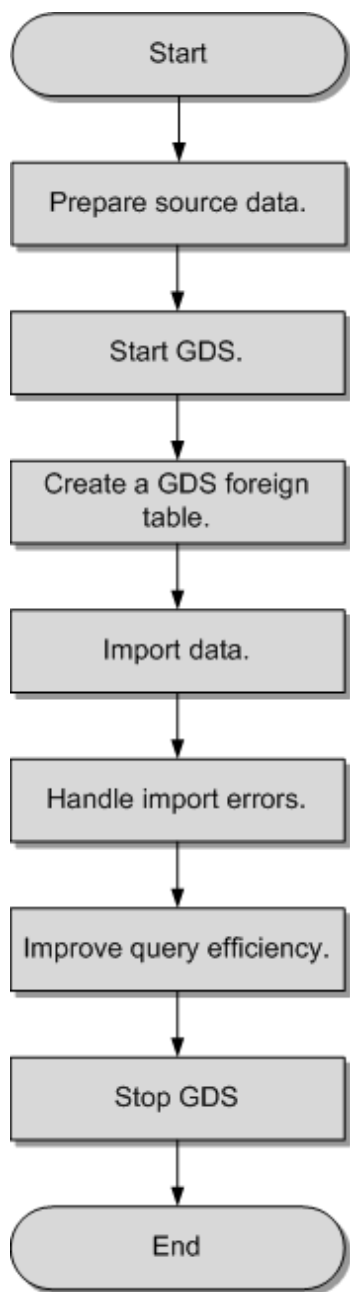


Table 2-13 Process description

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

2.3.2.2 Preparing Source Data

Scenario

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

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

Procedure

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

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

```
mkdir -p /input_data
```

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

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

----End

2.3.2.3 Installing, Configuring, and Starting GDS

Scenario

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

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

Background

The GDS version must match the cluster version. For example, GDS V100R008C00 matches DWS 1.3.X. Otherwise, the import or export may fail, or the import or export process may fail to respond. Therefore, use the latest version of GDS.

After the database is upgraded, download the latest version of DWS GDS as instructed in [Procedure](#). When the import or export starts, DWS checks the GDS versions. If the versions do not match, an error message is displayed and the import or export is terminated.

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

```
gds -V
```

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

```
SELECT version();
```

Procedure

Step 1 Log in as user **root** to the data server where GDS is to be installed and run the following command to create the directory for storing the GDS package:

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

Step 2 Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_8.x.x_suse_x64.zip** to the directory created in the previous step.

Step 3 (Optional) If SSL is used, upload the SSL certificates to the directory created in [Step 1](#).

Step 4 Go to the directory and decompress the package.

```
cd /opt/bin/dws
unzip dws_client_8.x.x_suse_x64.zip
```

Step 5 Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

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

Step 6 Change the owner of the GDS package directory and source data file directory to the GDS user.

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

Step 7 Switch to user **gds_user**.

```
su - gds_user
```

If the current cluster version is 8.0.x or earlier, skip [Step 8](#) and go to [Step 9](#).

If the current cluster version is 8.1.x, go to the next step.

Step 8 Execute the script on which the environment depends (applicable only to version 8.1.x).

```
cd /opt/bin/dws/gds/bin
source gds_env
```

Step 9 Start GDS.

GDS is eco-friendly software that can be launched once it is decompressed. You can start GDS in two ways: by using the **gds** command to configure startup parameters or by writing the parameters into the **gds.conf** configuration file and running the **gds_ctl.py** command to initiate GDS.

The first method is recommended when you do not need to import data again. The second method is recommended when you need to import data regularly.

- Method 1: Run the **gds** command to start GDS.

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

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

Example:

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

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

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

Example:

Run the following command to upload the SSL certificate mentioned in [Step 3](#) to **/opt/bin**:

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

Replace the information in italic as required.

- **-d *dir***: directory for storing data files that contain data to be imported. This tutorial uses **/input_data/** as an example.
- **-p *ip:port***: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with DWS. The port number ranges from 1024 to 65535.

The default port is **8098**. This tutorial uses **192.168.0.90:5000** as an example.

- **-H *address_string***: specifies the hosts that are allowed to connect to and use GDS. The value must be in CIDR format. Configure this parameter to enable a DWS cluster to access GDS for data import. Ensure that the network segment covers all hosts in a DWS cluster.
- **-l *log_file***: GDS log directory and log file name. This tutorial uses **/opt/bin/dws/gds/gds_log.txt** as an example.
- **-D**: GDS in daemon mode. This parameter is used only in Linux.
- **-t *worker_num***: number of concurrent GDS threads. The default value is **8**. The value is a positive integer ranging between 0 and 200 (included). If the data server and DWS have available I/O resources, you can increase the number of concurrent threads.

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

- **--enable-ssl**: enables SSL for data transmission.
- **--ssl-dir *Cert_file***: SSL certificate directory. Set this parameter to the certificate directory in [Step 3](#).

For details about GDS parameters, see "GDS - Parallel Data Loader > gds" in the *Data Warehouse Service (DWS) Tool Guide*.

- Method 2: Write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start GDS.
 - a. Run the following command to go to the **config** directory of the GDS package and modify the **gds.conf** configuration file. For details about the parameters in the **gds.conf** configuration file, see [Table 2-14](#).

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

Example:

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

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

Information in the configuration file is described as follows:

- The data server IP address is **192.168.0.90** and the GDS listening port is **5000**.
- Data files are stored in the **/input_data/** directory.
- Error log files are stored in the **/err** directory. The directory must be created by a user who has the GDS read and write permissions.
- The size of a single data file is 100 MB.
- The size of a single error log file is 100 MB.
- Logs are stored in the **/log/gds_log.txt** file. The directory must be created by a user who has the GDS read and write permissions.

- Only nodes with the IP address **10.10.0.*** can be connected.
 - The GDS process is running in daemon mode.
 - Recursive data file directories are used.
 - The number of concurrent import threads is 2.
- b. Start GDS and check whether it has been started.

```
python3 gds_ctl.py start
```

Example:

```
cd /opt/bin/dws/gds/bin
python3 gds_ctl.py start
Start GDS gds1 [OK]
gds [options]:
-d dir      Set data directory.
-p port     Set GDS listening port.
  ip:port   Set GDS listening ip address and port.
-l log_file Set log file.
-H secure_ip_range
            Set secure IP checklist in CIDR notation. Required for GDS to start.
-e dir      Set error log directory.
-E size     Set size of per error log segment.(0 < size < 1TB)
-S size     Set size of data segment.(1MB < size < 100TB)
-t worker_num Set number of worker thread in multi-thread mode, the upper limit is 200. If
without setting, the default value is 8.
-s status_file Enable GDS status report.
-D          Run the GDS as a daemon process.
-r          Read the working directory recursively.
-h          Display usage.
```

----End

gds.conf Parameter Description

Table 2-14 gds.conf configuration description

Attribute	Description	Value Range
name	Identifier	-
ip	Listening IP address	The IP address must be valid. Default value: 127.0.0.1
port	Listening port	Value range: 1024 to 65535 (integer) Default value: 8098
data_dir	Data file directory	-
err_dir	Error log file directory	Default value: data file directory
log_file	Log file Path	-
host	Host IP address allowed to be connected to GDS (The value must in CIDR format and this parameter is available for the Linux OS only.)	-

Attribute	Description	Value Range
recursive	Whether the data file directories are recursive Setting this to true will recursively read all files with the same name in the directory specified by location .	Value range: • true: recursive • false: not recursive Default value: false
daemon	Whether the process is running in daemon mode	Value range: • true: The process is running in daemon mode. • false: The process is not running in daemon mode. Default value: false
parallel	Number of concurrent data import threads	Value range: 0 to 200 (integer) Default value: 8

2.3.2.4 Creating a GDS Foreign Table

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

Procedure

Step 1 Collect source data information and GDS access information.

You need to collect the following source data information:

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

You need to collect the following GDS access information:

location: GDS URL. GDS information in [Installing, Configuring, and Starting GDS](#) is used as an example. In non-SSL mode, **location** is set to **gsfs://**

192.168.0.90:5000//input_data/. In SSL mode, **location** is set to **gsfss://192.168.0.90:5000//input_data/**. **192.168.0.90:5000** indicates the IP address and port number of GDS. **input_data** indicates the path of data source files managed by GDS. Replace the values as required.

Step 2 Design an error tolerance mechanism for data import.

DWS supports the following error tolerance in data import:

- **fill_missing_fields**: When the last column in a row of the source data file is empty, this parameter specifies whether to report an error or set this field in the row to **NULL**.

NOTE

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

- If this parameter is set to **true** or **on** and the last column of a data row in a source data file is lost, the column will be replaced with **null** and no error message will be generated.
- If this parameter is set to **false** or **off** and the last column of a data row in a source data file is lost, the following error information will be displayed:
missing data for column "tt"

Default value: **false** or **off**

- **ignore_extra_data**: When the number of columns in the source data file is greater than that specified in the foreign table, this parameter specifies whether to report an error or ignore the extra columns.

NOTE

Value range: true/on, false/off.

- When this parameter is **true** or **on** and the number of data source files exceeds the number of foreign table columns, excessive columns will be ignored.
- If the parameter is set to **false** or **off**, and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:
extra data after last expected column

Default value: **false** or **off**

- **per_node_reject_limit**: This parameter specifies the number of data format errors allowed on each DN. If the number of errors recorded in the error table on a DN exceeds the specified value, the import will fail and an error message will be reported. You can also set it to **unlimited**.
- **compatible_illegal_chars**: When an illegal character is encountered, this parameter specifies whether to import an error, or convert it and proceed with the import.

NOTE

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

- When the parameter is **true** or **on**, invalid characters are tolerated and imported to the database after conversion.
- If the parameter is **false** or **off**, and an error occurs when there are invalid characters, the import will be interrupted.

Default value: **false** or **off**

The following describes the rules for converting an invalid character:

- **\0** is converted to a space.

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

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

For example:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server
OPTIONS
(
  LOCATION 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
  FORMAT 'CSV' ,
  DELIMITER ',',
  ENCODING 'utf8',
  HEADER 'false',
  FILL_MISSING_FIELDS 'true',
  IGNORE_EXTRA_DATA 'true'
)
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

The following describes information in the preceding command:

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

----End

Example

For more examples, see [Example of Importing Data Using GDS](#).

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

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

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

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

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

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
FORMAT 'TEXT', delimiter E'\x08', null '', reject_limit '2', EOL '0x0D') WITH err_foreign_tpcds_reasons;
```

- Example 4: Create a GDS foreign table named **foreign_tpcds_reasons**. The data format is FIXED.

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk      integer      position(1,2),
  r_reason_id      char(16)     position(3,16),
  r_reason_desc    char(100)    position(19,100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/*', FORMAT 'FIXED', ENCODING
'utf8',FIX '119');
```

2.3.2.5 Importing Data

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

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

Prerequisites

The GDS server can communicate with the DWS cluster.

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

Procedure

Step 1 Create a table in DWS to store imported data.

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

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

```
DROP INDEX product_idx;
```

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

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

Step 3 Import data.

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

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

```
INSERT 0 9
```

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

NOTE

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

----End

Example:

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

```
CREATE TABLE reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
DISTRIBUTE BY HASH (r_reason_sk);
```

2. You are advised to delete the indexes from the target table before the import. Assume that an ordinary table index **reasons_idx** exists on the **r_reason_id** column in the **reasons** table. Delete the index before the import. Delete the index from the table.

```
DROP INDEX reasons_idx;
```

3. Import data from source data files through the **foreign_tpcds_reasons** foreign table to the **reasons** table.

```
INSERT INTO reasons SELECT * FROM foreign_tpcds_reasons ;
```

4. You can create indexes again after the import is complete.

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

2.3.2.6 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that arise during data import are categorized as either data format errors or non-data format errors. The error table records only data format errors.

- Data format error

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

```
SELECT * FROM error_table_name;
```

Table 2-15 lists the columns of the *error_table_name* table.

Table 2-15 Columns in the *error_table_name* table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where a data format error occurs If you use GDS for importing data, the error information includes the IP address and port number of the GDS server.
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error

If a non-data format error occurs during data import, the entire process is halted. During data import, you can identify and correct errors by reviewing the displayed error information.

Handling data import errors

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

Table 2-16 Handling data import errors

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

Error Information	Error Type	Cause	Solution
extra data after last expected column	Format error	The number of columns in the source data file is greater than that in the foreign table.	• Delete the unnecessary columns from the source data file. • When creating a foreign table, set the parameter ignore_extra_data to on. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.
invalid input syntax for type numeric: "a"	Format error	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to numeric .
null value in column "staff_id" violates not-null constraint	Non-format error	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the staff_id column.
duplicate key value violates unique constraint "reg_id_pk"	Non-format error	The unique constraint is violated.	• Delete the duplicate rows from the source data file. • Run the SELECT statement with the DISTINCT keyword to ensure that all imported rows are unique. INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;

Error Information	Error Type	Cause	Solution
value too long for type character varying(16)	Format error	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes (VARCHAR2).

2.3.2.7 Stopping GDS

Scenarios

Stop GDS after data is imported successfully.

Procedure

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

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

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

- a. Query the GDS process ID:

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

For example, the GDS process ID is 128954.

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

```
gds_user 128954 1 0 15:03 ? 00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -l /log/  
gds_log.txt -D  
gds_user 129003 118723 0 15:04 pts/0 00:00:00 grep gds
```

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

```
kill -9 128954
```

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

```
cd /opt/bin/dws/gds/bin  
python3 gds_ctl.py stop
```

----End

2.3.2.8 Example of Importing Data Using GDS

Parallel Import from Multiple Data Servers

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

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

```
mkdir -p /input_data
```
2. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group exist, skip this step.

```
groupadd gdsgrp  
useradd -g gdsgrp gds_user
```
3. Evenly distribute CSV source data files to the **/input_data** directories on the data servers.
4. Change the owners of source data files and the **/input_data** directory on each data server to **gds_user**. The data server whose IP address is **192.168.0.90** is used as an example.

```
chown -R gds_user:gdsgrp /input_data
```
5. Log in to each data server as user **gds_user** and start GDS.
The GDS installation path is **/opt/bin/dws/gds**. Source data files are stored in **/input_data/**. The IP addresses of the data servers are **192.168.0.90** and **192.168.0.91**. The GDS listening port is **5000**. GDS runs in daemon mode.
Start GDS on the data server whose IP address is **192.168.0.90**.

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


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

```
/opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.91:5000 -H 10.10.0.1/24 -D
```
6. Use a tool to connect to the database.
7. Create the target table **tpcds.reasons**.

```
CREATE TABLE tpcds.reasons  
(  
  r_reason_sk integer not null,  
  r_reason_id char(16) not null,  
  r_reason_desc char(100)  
);
```
8. Create the foreign table **tpcds.foreign_tpcds_reasons** for receiving data from the data server.
Data export mode settings are as follows:
 - Set the import mode to **Normal**.
 - When GDS is started, the source data file directory is **/input_data** and the GDS listening port is **5000**. Therefore, set **location** to **gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/***.Information about the data format is configured based on data format parameters specified during data export. The parameter configurations are as follows:
 - **format** is set to **CSV**.
 - **encoding** is set to **UTF-8**.
 - **delimiter** is set to **E'\x08'**.
 - **quote** is set to **E'\x1b'**.
 - **null** is set to an empty string without quotation marks.
 - **escape** defaults to the value of **quote**.
 - **header** is set to **false**, indicating that the first row is identified as a data row in an imported file.

Configure import error tolerance parameters as follows:

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

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

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

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

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

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

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

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

Data Import Using Multiple Threads

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

1. Log in to the GDS data server as user **root**, and then create the data file directory **/input_data** and its sub-directories **/input_data/import1/** and **/input_data/import2/**.

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

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

```
chown -R gds_user:gdsgrp /input_data
```
5. Log in to the data server as user **gds_user** and start GDS.

The GDS installation path is **/opt/bin/dws/gds**. Source data files are stored in **/input_data/**. The IP address of the data server is **192.168.0.90**. The GDS

listening port is **5000**. GDS runs in daemon mode. The degree of parallelism is 2. A recursive directory is specified.

```
/opt/bin/dws/gds/bin/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D -t 2 -r
```

6. Use a tool to connect to the database.
7. In the database, create the target tables **tpcds.reasons1** and **tpcds.reasons2**.

```
CREATE TABLE tpcds.reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
CREATE TABLE tpcds.reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
```

8. In the database, create the foreign tables **tpcds.foreign_tpcds_reasons1** and **tpcds.foreign_tpcds_reasons2** for the source data.

The foreign table **tpcds.foreign_tpcds_reasons1** is used as an example to describe how to configure parameters in a foreign table.

Data export mode settings are as follows:

- Set the import mode to **Normal**.
- When GDS is started, the configured source data file directory is **/input_data** and the GDS listening port is **5000**. However, source data files are actually stored in **/input_data/import1/**. Therefore, set **location** to **gsfs://192.168.0.90:5000/import1/***.

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

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **E'\x1b'**.
- **null** is set to an empty string without quotation marks.
- **escape** defaults to the value of **quote**.
- **header** is set to **false**, indicating that the first row is identified as a data row in an imported file.

Configure import error tolerance parameters as follows:

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

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

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons1
(
```

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

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

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

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

```
INSERT INTO tpcds.reasons1 SELECT * FROM tpcds.foreign_tpcds_reasons1;
INSERT INTO tpcds.reasons2 SELECT * FROM tpcds.foreign_tpcds_reasons2;
```
- Query data import errors in the **err_tpcds_reasons1** and **err_tpcds_reasons2** tables and rectify the errors (if any). For details, see [Handling Import Errors](#).

```
SELECT * FROM err_tpcds_reasons1;
SELECT * FROM err_tpcds_reasons2;
```
- After data import is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is **128954**.

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

Importing Data Through a Pipe File

Step 1 Start GDS.

```
/opt/bin/dws/gds/bin/gds -d /***gds_data/ -D -p 192.168.0.1:7789 -l /***gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Import data.

- Log in to the database and create an internal table.

```
CREATE TABLE test_pipe_1( id integer not null, gender text not null, name text );
```
- Create a read-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',', NULL '',
EOL '0x0a', file_type 'pipe', auto_create_pipe 'false');
```
- Execute the import statement and the statement will be blocked.

```
INSERT INTO test_pipe_1 SELECT * FROM foreign_test_pipe_tr;
```

Step 3 Import data through the GDS pipes.

- Log in to GDS and go to the GDS data directory.

```
cd /***gds_data/
```
- Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.

```
mkfifo foreign_test_pipe.pipe;
```

 NOTE

A pipe will be automatically cleared after an operation is complete. To perform another operation, create a pipe file again.

3. Write data to the pipe.
`cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe`
4. To read the compressed file to the pipe, run the following command.
`gzip -d < out.gz > foreign_test_pipe.pipe`
5. To read the HDFS file to the pipe, run the following command.
`hdfs dfs -cat - /user/hive/***/test_pipe.txt > foreign_test_pipe.pipe`

Step 4 View the result returned by the import statement.

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | gender |   name
----+-----+-----
3 | 2 | 1111111111111111
1 | 2 | 1111111111111111
2 | 2 | 1111111111111111
4 | 2 | 1111111111111111
(4 rows)
```

----End

Importing Data Through Multi-Process Pipes

GDS also supports importing data through multi-process pipes. That is, one foreign table corresponds to multiple GDSs.

The following takes importing a local file as an example.

Step 1 Start multiple GDSs. If the GDSs have been started, skip this step.

```
/opt/bin/dws/gds/bin/gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
/opt/bin/dws/gds/bin/gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log_1/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Import data.

1. Log in to the database and create an internal table.
`CREATE TABLE test_pipe( id integer not null, gender text not null, name text );`
2. Create a read-only foreign table.
`CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/foreign_test_pipe.pipe|gsfs://192.168.0.1:7790/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a', file_type 'pipe', auto_create_pipe 'false');`
3. Execute the export statement and the statement will be blocked.
`INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;`

Step 3 Import data through the GDS pipes.

1. Log in to GDS and go to each GDS data directory.
`cd /***/gds_data/`
`cd /***/gds_data_1/`
2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
`mkfifo foreign_test_pipe.pipe;`
3. Read each pipe and write the new file to the pipes.

```
cat postgres_public_foreign_test_pipe_tw.txt > foreign_test_pipe.pipe
```

Step 4 View the result returned by the import statement.

```
INSERT INTO test_pipe_1 select * from foreign_test_pipe_tr;
INSERT 0 4
SELECT * FROM test_pipe_1;
id | gender |   name
----+-----+-----
3 | 2 | 1111111111111111
1 | 2 | 1111111111111111
2 | 2 | 1111111111111111
4 | 2 | 1111111111111111
(4 rows)
```

----End

Direct Data Import Between Clusters

Step 1 Start the GDS. (If the process has been started, skip this step.)

```
gds -d /***/gds_data/ -D -p GDS_IP:GDS_PORT -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data from the source database.

1. Log in to the target database, create an internal table, and write data to the table.

```
CREATE TABLE test_pipe( id integer not null, gender text not null, name text );
INSERT INTO test_pipe values(1,2,'1111111111111111');
INSERT INTO test_pipe values(2,2,'1111111111111111');
INSERT INTO test_pipe values(3,2,'1111111111111111');
INSERT INTO test_pipe values(4,2,'1111111111111111');
INSERT INTO test_pipe values(5,2,'1111111111111111');
```

2. Create a write-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe( id integer not null, age text not null, name text ) SERVER
gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/', FORMAT 'text', DELIMITER ',', NULL '',
EOL '\0x0a', file_type 'pipe') WRITE ONLY;
```

3. Execute the import statement. The statement is blocked.

```
INSERT INTO foreign_test_pipe SELECT * FROM test_pipe;
```

Step 3 Import data to the target cluster.

1. Create an internal table.

```
CREATE TABLE test_pipe (id integer not null, gender text not null, name text);
```

2. Create a read-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server OPTIONS (LOCATION
'gsfs://GDS_IP:GDS_PORT/', FORMAT 'text', DELIMITER ',', NULL '', EOL '\0x0a', file_type 'pipe',
auto_create_pipe 'false');
```

3. Run the following command to import data to the table.

```
INSERT INTO test_pipe SELECT * FROM foreign_test_pipe;
```

Step 4 View the result returned by the import statement from the target cluster.

```
SELECT * FROM test_pipe;
id | gender |   name
----+-----+-----
3 | 2 | 1111111111111111
6 | 2 | 1111111111111111
7 | 2 | 1111111111111111
1 | 2 | 1111111111111111
2 | 2 | 1111111111111111
4 | 2 | 1111111111111111
5 | 2 | 1111111111111111
```



```
8 | 2 | 11111111111111111111
9 | 2 | 11111111111111111111
(9 rows)
```

----End

NOTE

By default, the pipeline file exported from or imported to GDS is named in the format of *Database name_Schema name_Foreign table name .pipe*. Therefore, the database name and schema name of the target cluster must be the same as those of the source cluster. If the database or schema is inconsistent, you can specify the same pipe file in the URL of the **location**.

Example:

- Pipe name specified by a write-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe(id integer not null, age text not null, name text)
SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe',
FORMAT 'text', DELIMITER ',', NULL '', EOL '\000a', file_type 'pipe') WRITE ONLY;
```
- Pipe name specified by a read-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe(like test_pipe) SERVER gsmpp_server OPTIONS
(LOCATION 'gsfs://GDS_IP:GDS_PORT/foreign_test_pipe.pipe', FORMAT 'text', DELIMITER ',', NULL
'', EOL '\000a', file_type 'pipe', auto_create_pipe 'false');
```

2.3.3 Importing Data from MRS to a Cluster

2.3.3.1 Overview

MRS is a big data cluster running based on the open-source Hadoop ecosystem. It provides the industry's latest cutting-edge storage and analytical capabilities of massive volumes of data, satisfying your data storage and processing requirements. For details, see the *MapReduce Service User Guide*.

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

NOTICE

Ensure that MRS can communicate with DWS:

Scenario 1: If MRS and DWS are in the same region and VPC, they can communicate with each other by default.

Scenario 2: If MRS and DWS are in the same region but in different VPCs, you need to create a VPC peering connection. For details, see "VPC Peering Connection Overview" in *Virtual Private Cloud User Guide*.

Scenario 3: If MRS and DWS are not in the same region, you need to use Cloud Connect (CC) to create network connections. For details, see the user guide of the corresponding service.

Scenario 4: If MRS is deployed on-premises, you need to use Direct Connect (DC) or Virtual Private Network (VPN) to connect the network. For details, see the User Guide of the corresponding service.

Importing Data from MRS to a DWS Cluster

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

2.3.3.2 Preparing Data in an MRS Cluster

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

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

If you have completed the preparations, skip this section.

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

Data File

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

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

230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon
Shirt,black,M,2056,2017-10-02,3842,very good

Creating a Hive ORC Table in an MRS Cluster

1. Create an MRS cluster.

For details, see "Creating a Cluster > Creating a User-Defined Cluster" in the *MapReduce Service User Guide*.

2. Download the client.

- a. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that EIP needs to be bound, bind an EIP first.
- b. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.
- c. Choose **Cluster > Name of the desired cluster > Dashboard**. On the page that is displayed, choose **More > Download Client**. The **Download Cluster Client** dialog box is displayed.

Download Cluster Client

Download the  client. The cluster client provides all services.

Select Client Type: ☒ Complete Client ☐ Configuration Files Only

Select Platform Type: ☒ x86_64 ☐ aarch64

☐ Save to Path: 

OK

Cancel

NOTE

To obtain the client of an earlier version, choose **Services > Download Client** and set **Select Client Type** to **Configuration Files Only**.

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

- a. Log in to a Master node.

For details, see "Logging in to a cluster > Logging In to an ECS" in the *MapReduce Service User Guide*.

- b. Switch the user.

```
sudo su - omm
```

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

```
cd /opt/client
```

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

```
source bigdata_env
```

- e. If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission to create Hive tables.

For details, see section "Creating a Role" in the *MapReduce Service User Guide*.

- f. Configure roles with the corresponding permissions.

For details, see section "Creating a User" in the *MapReduce Service User Guide*.

- g. Bind roles to users. If the Kerberos authentication is disabled for the current cluster, skip this step.

```
kinit MRS cluster user
```

Example: **kinit hiveuser**

- h. Run the following command to start the Hive client:

```
beeline
```

4. Create a database demo on Hive.

Run the following command to create the database demo:

```
CREATE DATABASE demo;
```

5. Create table **product_info** of the **Hive TEXTFILE** type in the database demo and import the **Data File** (**product_info.txt**) to the HDFS path corresponding to the table.

Run the following command to switch to the database demo:

```
USE demo;
```

Run the following command to create table **product_info** and define the table fields based on data in the **Data File**.

```
DROP TABLE product_info;
```

```
CREATE TABLE product_info
```

```
(
```

```
  product_price      int      ,
  product_id         char(30) ,
  product_time       date     ,
  product_level      char(10) ,
  product_name       varchar(200) ,
  product_type1      varchar(20) ,
  product_type2      char(10)  ,
  product_monthly_sales_cnt int  ,
  product_comment_time date    ,
  product_comment_num int     ,
  product_comment_content varchar(200)
```

```
)
```

```
row format delimited fields terminated by ','
stored as TEXTFILE;
```

For details about how to import data to an MRS cluster, see "Cluster Operation Guide > Managing Active Clusters > Managing Data Files" in the *MapReduce Service User Guide*.

6. Create a Hive ORC table named **product_info_orc** in the database demo.

Run the following command to create the Hive ORC table **product_info_orc**. The table fields are the same as those of the **product_info** table created in the previous step.

```
DROP TABLE product_info_orc;

CREATE TABLE product_info_orc
(
  product_price      int      ,
  product_id         char(30) ,
  product_time       date     ,
  product_level      char(10) ,
  product_name       varchar(200) ,
  product_type1      varchar(20) ,
  product_type2      char(10)  ,
  product_monthly_sales_cnt int ,
  product_comment_time date    ,
  product_comment_num int      ,
  product_comment_content varchar(200)
)
row format delimited fields terminated by ','
stored as orc;
```

7. Insert data in the **product_info** table to the Hive ORC table **product_info_orc**.
`INSERT INTO product_info_orc SELECT * FROM product_info;`

Query table **product_info_orc**.

```
SELECT * FROM product_info_orc;
```

If data displayed in the **Data File** can be queried, the data has been successfully inserted to the ORC table.

2.3.3.3 Manually Creating a Foreign Server

In the syntax **CREATE FOREIGN TABLE (SQL on Hadoop or OBS)** for creating a foreign table, you need to specify a foreign server associated with the MRS data source connection.

When you create an MRS data source connection on the DWS console, the database administrator **dbadmin** automatically creates a foreign server in the default database **postgres**. If you want to create a foreign table in the default database **postgres** to read MRS data, skip this section.

To allow a common user to create a foreign table in a user-defined database to read MRS data, you must manually create a foreign server in the user-defined database. This section describes how does a common user create a foreign server in a user-defined database. The procedure is as follows:

1. Ensure that an MRS data source connection has been created for the DWS cluster.
For details, see section "MRS Data Sources > Creating an MRS Data Source Connection" in the *Data Warehouse Service User Guide*.
2. [Creating a User and a Database and Granting the User Foreign Table Permissions](#)
3. [Manually Creating a Foreign Server](#)

 NOTE

If you no longer need to read data from the MRS data source and have deleted the MRS data source on the DWS console, only the foreign server automatically created in the default database **postgres** will be deleted, and the manually created foreign server needs to be deleted manually. For details about how to delete a foreign server, see [Deleting the Manually Created Foreign Server](#).

Creating a User and a Database and Granting the User Foreign Table Permissions

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

- Step 1** Connect to the default database **gaussdb** as a database administrator through the database client tool provided by DWS.

For example, use the **gsq** client to connect to the database by running the following command:

```
gsq -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

- Step 2** Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

Switch to the created user.

```
SET ROLE dbuser PASSWORD 'password';
```

Run the following command to create a database:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

datname	datdba	encoding	datcollate	datctype	datistemplate	datallowconn	datconnlimit	datacl
template1	10	0	C	C	t	t	-1	14146 1351
1663	ORA		{=c/Ruby,Ruby=CTc/Ruby}					
template0	10	0	C	C	t	f	-1	14146 1350
1663	ORA		{=c/Ruby,Ruby=CTc/Ruby}					
gaussdb	10	0	C	C	f	t	-1	14146 1352
1663	ORA		{=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,huobinru=C/Ruby}					
mydatabase	17000	0	C	C	f	t	-1	14146 1351
1663	ORA							

(4 rows)

- Step 3** Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Use the connection to create a database as a database administrator.

You can use the **gsql** client to run the following command, switching to an administrator user, and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password as prompted.

NOTE

Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to grant the user the permission to use foreign tables:

```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
       r.rolcreatorole, r.rolcreatedb, r.rolcanlogin,
       r.rolconlimit, r.rolvalidbegin, r.rolvaliduntil,
       ARRAY(SELECT b.rolname
              FROM pg_catalog.pg_auth_members m
              JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
              WHERE m.member = r.oid) as memberof
       , r.rolreplication
       , r.rolauditadmin
       , r.rolsystemadmin
       , r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the **UseFT** permission.

rolname	rolsuper	rolinherit	rolcreatorole	rolcreatedb	rolcanlogin	rolconlimit	rolvalidbegin	rolvaliduntil	memberof	rolreplication	rolauditadmin	rolsystemadmin	roluseft
dbuser	f	t	f	t	t	-1			{}	f			
lily	f	t	f	f	t	-1			{}	f			
Ruby	t	t	t	t	t	-1			{}	t			

----End

Manually Creating a Foreign Server

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by DWS.

You can use the **gsql** client to log in to the database in either of the following ways:

You can use either of the following methods to create the connection:

- If you have logged in to the gsql client, run the following command to switch the database and user:

```
\c postgres dbadmin;
```


Enter the password as prompted.
- If you have not logged in to the gsql client or have exited the gsql client by running the `\q` command, run the following command to reconnect to it:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

Step 2 Run the following command to query the information about the foreign server that is automatically created:

```
SELECT * FROM pg_foreign_server;
```

The returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
srvoptions					
gsmpp_server	10	13673			
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca	16476	13685			
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}					
(2 rows)					

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

Step 3 Switch to the user who is about to create a foreign server to connect to the corresponding database.

In this example, run the following command to use common user **dbuser** created in [Creating a User and a Database and Granting the User Foreign Table Permissions](#) to connect to **mydatabase** created by the user:

```
\c mydatabase dbuser;
```

Step 4 Create a foreign server.

For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*. For example:

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca FOREIGN DATA  
WRAPPER HDF5_FDW  
OPTIONS  
(  
  address '192.168.1.245:25000,192.168.1.218:25000',  
  hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',  
  type 'hdfs'  
);
```

Mandatory parameters are described as follows:

- *Name of the foreign server*
You can customize a name.
In this example, specify the name to the value of the **srvname** field recorded in [Step 2](#), such as `hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca`.
Resources in different databases are isolated. Therefore, the names of foreign servers in different databases can be the same.
- **FOREIGN DATA WRAPPER**
This parameter can only be set to **HDFS_FDW**, which already exists in the database.
- **OPTIONS** parameters
Set the following parameters to the values under **srvoptions** recorded in [Step 2](#).
 - **address**
Specifies the IP address and port number of the primary and standby nodes of the HDFS cluster.
 - **hdfscfgpath**
Specifies the configuration file path of the HDFS cluster. This parameter is available only when **type** is **HDFS**. You can set only one path.
 - **type**
Its value is **hdfs**, which indicates that **HDFS_FDW** connects to HDFS.

Step 5 View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE  
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

The server is successfully created if the returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca	16476	13685			

```
{ "address=192.168.1.245:25000,192.168.1.218:25000",  
  "hdfscfgpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca",  
  "type=hdfs" }
```

(1 row)

----End

2.3.3.4 Creating a Foreign Table

This section describes how to create a Hadoop foreign table in the DWS database to access the Hadoop structured data stored on MRS HDFS. A Hadoop foreign table is read-only. It can only be queried using **SELECT**.

Prerequisites

- You have created an MRS cluster and imported data to the ORC table in the Hive/Spark database.
For details, see [Preparing Data in an MRS Cluster](#).
- You have created an MRS data source connection for the DWS cluster.
For details, see section "MRS Data Sources > Creating an MRS Data Source Connection" in the *Data Warehouse Service User Guide*.

Obtaining the HDFS Path of the MRS Data Source

There are two methods for you to obtain the HDFS path.

- **Method 1**

For Hive data, log in to the Hive client of MRS (see [2](#)), run the following command to view the detailed information about the table, and record the data storage path in the **location** parameter:

```
use <database_name>;  
desc formatted <table_name>;
```

For example, if the value of the **location** parameter in the returned result is **hdfs://hacluster/user/hive/warehouse/demo.db/product_info_orc/**, the HDFS path is **/user/hive/warehouse/demo.db/product_info_orc/**.

- **Method 2**

Perform the following steps to obtain the HDFS path:

- a. Log in to the MRS console.
- b. Choose **Cluster > Active Cluster** and click the name of the cluster to be queried to enter the page displaying the cluster's basic information.
- c. Click **File Management** and select **HDFS File List**.
- d. Go to the storage directory of the data to be imported to the DWS cluster and record the path.

Obtaining Information About the Foreign Server Connected to the MRS Data Source

Step 1 Use the user who creates the foreign server to connect to the corresponding database.

Determine whether to use a common user to create a foreign table in the customized database based on requirements.

- **Yes**

- a. Ensure that you have created the common user **dbuser** and its database **mydatabase**, and manually created a foreign server in **mydatabase** by following steps in [Manually Creating a Foreign Server](#).
- b. Connect to the database **mydatabase** as user **dbuser** through the database client tool provided by DWS.

If you have connected to the database using the **gsql** client, run the following command to switch the user and database:

```
\c mydatabase dbuser;
```

Enter your password as prompted.

- **No**

When you create an MRS data source connection on the DWS console, the database administrator **dbadmin** automatically creates a foreign server in the default database **postgres**. If you create a foreign table in the default database **postgres** as the database administrator **dbadmin**, you need to connect to the database using the database client tool provided by DWS. For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

Step 2 Run the following command to view the information about the created foreign server connected to the MRS data source:

```
SELECT * FROM pg_foreign_server;
```

NOTE

You can also run the `\desc+` command to view the information about the foreign server.

The returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
srvoptions					
gsmpp_server	10	13673			
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca			16476	13685	
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}					
(2 rows)					

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

----End

Creating a Foreign Table

After [Obtaining Information About the Foreign Server Connected to the MRS Data Source](#) and [Obtaining the HDFS Path of the MRS Data Source](#) are completed, you can create a foreign table to read data from the MRS data source.

The syntax for creating a foreign table is as follows. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
    column_constraint [...] } ]
  table_constraint [, ...] [, ...] } )
SERVER dfs_server
OPTIONS ( { option_name 'value' } [, ...] )
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *foreign_product_info*, set parameters in the syntax as follows:

- **table_name**
Mandatory. This parameter specifies the name of the foreign table to be created.

- Table column definitions

- **column_name**: specifies the name of a column in the foreign table.
- **type_name**: specifies the data type of the column.

Multiple columns are separate by commas (,).

The number of columns and column types in the foreign table must be the same as those in the data stored on MRS. Learn [Data Type Conversion](#) before defining column data types.

- **SERVER dfs_server**

This parameter specifies the foreign server name of the foreign table. This server must exist. The foreign table can read data from an MRS cluster by configuring the foreign server and connecting to the MRS data source.

Enter the value of the **srvname** field queried in [Obtaining Information About the Foreign Server Connected to the MRS Data Source](#).

- **OPTIONS** parameters

These are parameters associated with the foreign table. The key parameters are as follows:

- **format**: This parameter is mandatory. The value can only be **orc**. It specifies the format of the source data file. Only Hive ORC files are supported.
- **foldername**: This parameter is mandatory. It specifies the HDFS directory for storing data or data file path.

If the MRS analysis cluster has enabled Kerberos authentication, ensure that the MRS user having the MRS data source connection has the read and write permissions for the directory.

Follow the steps in [Obtaining the HDFS Path of the MRS Data Source](#) to obtain the HDFS path, which is the value of parameter **foldername**.

- **encoding**: This parameter is optional. It specifies the encoding format of a source data file in the foreign table. Its default value is **utf8**.
- **DISTRIBUTE BY**

This parameter specifies the data read mode for the foreign table. There are two read modes supported. In this example, **ROUNDROBIN** is selected.

- **ROUNDROBIN**: When a foreign table reads data from the data source, each node in a DWS cluster randomly reads some data and integrates the random data to a complete data set.
 - **REPLICATION**: When a foreign table reads data from the data source, each node in the DWS cluster reads a complete data set.
 - Other parameters in the syntax
- Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

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

```
DROP FOREIGN TABLE IF EXISTS foreign_product_info;  
  
CREATE FOREIGN TABLE foreign_product_info  
(
```

```
product_price      integer      ,
product_id         char(30)    ,
product_time       date        ,
product_level      char(10)    ,
product_name       varchar(200),
product_type1      varchar(20) ,
product_type2      char(10)    ,
product_monthly_sales_cnt integer ,
product_comment_time date      ,
product_comment_num integer    ,
product_comment_content varchar(200)
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
format 'orc',
encoding 'utf8',
foldername '/user/hive/warehouse/demo.db/product_info_orc/'
)
DISTRIBUTE BY ROUNDROBIN;
```

Data Type Conversion

Data is imported to Hive/Spark and then stored on HDFS in ORC format. Actually, DWS reads ORC files on HDFS, and queries and analyzes data in these files.

Data types supported by Hive/Spark are different from those supported by DWS. Therefore, you need to learn the mapping between them. [Table 2-17](#) describes the mapping in detail.

Table 2-17 Data type mapping

Type	Column Type Supported by an HDFS/OBS Foreign Table of DWS	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
Integer in two bytes	SMALLINT	SMALLINT	SMALLINT
Integer in four bytes	INTEGER	INT	INT
Integer in eight bytes	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4 (REAL)	FLOAT	FLOAT
Double-precision floating point number	FLOAT8(DOUBLE PRECISION)	DOUBLE	FLOAT
Scientific data type	DECIMAL[p (,s)] The maximum precision can reach up to 38.	DECIMAL The maximum precision can reach up to 38 (Hive 0.11).	DECIMAL

Type	Column Type Supported by an HDFS/OBS Foreign Table of DWS	Column Type Supported by a Hive Table	Column Type Supported by a Spark Table
Date type	DATE	DATE	DATE
Time type	TIMESTAMP	TIMESTAMP	TIMESTAMP
BOOLEAN type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n)	CHAR (n)	STRING
VARCHAR type	VARCHAR(n)	VARCHAR (n)	VARCHAR (n)
String	TEXT(CLOB)	STRING	STRING

2.3.3.5 Importing Data

Viewing Data in the MRS Data Source by Directly Querying the Foreign Table

If the data amount is small, you can directly run SELECT to query the foreign table and view the data in the MRS data source.

Step 1 Run the following command to query data from the foreign table:

```
SELECT * FROM foreign_product_info;
```

If the query result is the same as the data in [Data File](#), the import is successful. The following information is displayed at the end of the query result:

```
(20 rows)
```

After data is queried, you can insert the data to common tables in the database.

----End

Querying Data After Importing It

You can query the MRS data after importing it to DWS.

Step 1 Create a table in DWS to store imported data.

The target table structure must be the same as the structure of the foreign table created in [Creating a Foreign Table](#). That is, both tables must have the same number of columns and column types.

For example, create a table named **product_info**. The table example is as follows:

```
DROP TABLE IF EXISTS product_info;  
CREATE TABLE product_info  
(  
    product_price      integer      ,
```

```
product_id      char(30)      ,
product_time    date          ,
product_level   char(10)     ,
product_name    varchar(200) ,
product_type1   varchar(20)  ,
product_type2   char(10)     ,
product_monthly_sales_cnt integer ,
product_comment_time date    ,
product_comment_num integer ,
product_comment_content varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

Step 2 Run the **INSERT INTO .. SELECT ..** command to import data from the foreign table to the target table.

Example:

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

If information similar to the following is displayed, the data has been imported.
INSERT 0 20

Step 3 Run the following **SELECT** command to view data imported from MRS to DWS:

```
SELECT * FROM product_info;
```

If the query result is the same as the data in [Data File](#), the import is successful. The following information is displayed at the end of the query result:

```
(20 rows)
```

```
----End
```

Error Handling

During data import, the following error information indicates that DWS is to read an ORC data file but the actual file is in TXT format. Therefore, create a table of the Hive ORC type by referring to [Creating a Hive ORC Table in an MRS Cluster](#) and store the data to the table.

```
ERROR: dn_6009_6010: Error occurs while creating an orc reader for file /user/hive/warehouse/
products_info.txt, detail can be found in dn log of dn_6009_6010.
```

2.3.3.6 Deleting Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation.

Deleting the Foreign Table and Target Table

Step 1 (Optional) If operations in [Querying Data After Importing It](#) have been performed, run the following command to delete the target table:

```
DROP TABLE product_info;
```

Step 2 Run the following command to delete the foreign table:

```
DROP FOREIGN TABLE foreign_product_info;
```

----End

Deleting the Manually Created Foreign Server

If operations in [Manually Creating a Foreign Server](#) have been performed, perform the following steps to delete the foreign server, database, and user:

Step 1 Use the client provided by DWS to connect to the database where the foreign server is located as the user who created the foreign server.

You can use the **gsql** client to log in to the database in either of the following ways:

- If you have logged in to the gsql client, run the following command to switch the database and user:

```
\c mydatabase dbuser;
```

Enter the password as prompted.

- If you have logged in to the gsql client, you can run the **\q** command to exit gsql, and run the following command to reconnect to it:

```
gsql -d mydatabase -h 192.168.2.30 -U dbuser -p 8000 -r
```

Enter the password as prompted.

Step 2 Delete the manually created foreign server.

Run the following command to delete it:

```
DROP SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca;
```

The foreign server is deleted if the following information is displayed:

```
DROP SERVER
```

View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE  
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

The server is successfully deleted if the returned result is as follows:

```
srvname | srvowner | srvfwdw | srvtype | srversion | srvacl | srvoptions  
-----+-----+-----+-----+-----+-----+-----  
(0 rows)
```

Step 3 Delete the customized database.

Connect to the default database **postgres** through the database client tool provided by DWS.

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c postgres
```

Enter your password as prompted.

Run the following command to delete the customized database:

```
DROP DATABASE mydatabase;
```

The database is deleted if the following information is displayed:


```
DROP DATABASE
```

Step 4 Delete the common user created in this example as the administrator.

Connect to the database as a database administrator through the database client tool provided by DWS.

If you have logged in to the database using the **gsql** client, run the following command to switch the database and user:

```
\c postgres dbadmin
```

Run the following command to reclaim the permission for creating foreign servers:
REVOKE ALL ON FOREIGN DATA WRAPPER hdfs_fdw FROM dbuser;

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to delete the user:

```
DROP USER dbuser;
```

You can run the **\du** command to query for the user and check whether the user has been deleted.

----End

2.3.4 GDS-based Cross-Cluster Interconnection

Function

With data processing based on foreign tables, GDS is used to transfer data and synchronize data between multiple clusters.

Scenarios

- Data is synchronized from one cluster to another. Full data synchronization and data synchronization based on filter criteria are supported.
- Currently, only the following syntax is supported.
 - `INSERT INTO inner table SELECT... FROM linked_external_table 1 [WHERE];`
 - `INSERT INTO linked_external_table SELECT * FROM inner table 1 [JOIN inner table 2 | WHERE];`
 - `SELECT ... FROM linked_external_table;`

Important Notes

- The column name and type of the created foreign table must be the same as those of the corresponding source table, and the tables in the remote cluster must be row-store or column-store.
- Before running the synchronization statement, ensure that the tables to be synchronized exist in the local and remote clusters.
- The status of the two clusters is **Normal**.
- Both clusters must be able to connect to each other using GDS.
- The database code of both clusters must be the same. Otherwise, an error may be reported or the received data may be garbled characters.

- The compatible database types specified for both clusters must be the same. Otherwise, an error may be reported or the received data may be garbled characters.
- Ensure that the user performing table synchronization has the permission to access those tables.
- Foreign tables for interconnection can be used only for cross-cluster data synchronization. In other scenarios, errors may occur or the operation may be invalid.
- The foreign tables for interconnection do not support complex column expressions or complex syntax, including **join**, **sort**, **cursor**, **with**, and **set**.
- SQL statements that are not pushed down cannot use this feature to synchronize data.
- The explain plan and logical cluster are not supported.
- Only the simple JDBC mode is supported.
- If data is synchronized from the local cluster to a remote cluster, only internal table query is supported.
- The GDS specified by the **syncsrv** option of foreign servers does not support the SSL mode.
- After data synchronization is complete, only the number of data rows is verified.
- The maximum number of concurrent services cannot be greater than half of the value of the GDS startup parameter **-t** and cannot be greater than the value of **max_active_statements**. Otherwise, services may fail due to timeout.

Preparations

- Configure the interconnection between both clusters.
- Plan and deploy GDS servers. Ensure that all GDS servers can communicate with all nodes in the two clusters. That is, the security group of the GDS servers must allow the corresponding GDS port (for example, 5000) and DWS port (8000 by default) in the inbound direction. For details about how to deploy GDS, see [Installing, Configuring, and Starting GDS](#).

NOTE

When starting GDS, you can specify any directory as the data transit directory, for example, **/opt**. An example of the startup command is as follows:

```
/opt/gds/bin/gds -d /opt -p 192.168.0.2:5000 -H 192.168.0.1/24 -l /opt/gds/bin/gds_log.txt -D -t 2
```

Procedure

Assume that the table **tbl_remote** in the remote cluster is to be synchronized with the table **tbl_local** in the local cluster and the user performing the synchronization is **user_remote**. Note that the user must have the permission to access the **tbl_remote** table.

Step 1 Create a server.

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS(  
  address '192.168.178.207:8000',  
  dbname 'db_remote',
```

```
username 'user_remote',  
password 'xxxxxxx',  
syncsrv 'gsfs://192.168.178.129:5000|gsfs://192.168.178.129:5000'  
);
```

- **server_remote** indicates the server name, which is used by the foreign table for interconnection.
- **address** indicates the IP address and port number of the CN in the remote cluster. Only one address is allowed.
- **dbname** indicates the database name of the remote cluster.
- **username** indicates the username used for connecting to the remote cluster. This user cannot be a system administrator.
- **password** indicates the password used for connecting to the remote cluster.
- **syncsrv** indicates the IP address and port number of the GDS server. If there are multiple addresses, use vertical bars (|) to separate them. The function of **syncsrv** is similar to that of **location** in the GDS foreign table.

NOTE

DWS tests the network connected to the GDS addresses set by **syncsrv**.

- The test can only show the network status between the local cluster and the GDSs, but cannot show the network status between the remote cluster and GDS. You need to check the error message.
- After removing the unavailable GDSs, select a proper number of GDSs that do not cause service suspension to synchronize data.

Step 2 Create a foreign table for interconnection.

```
CREATE FOREIGN TABLE ft_tbl(  
col_1 type_name,  
col_2 type_name,  
...  
) SERVER server_remote OPTIONS (  
schema_name 'schema_remote',  
table_name 'tbl_remote',  
encoding 'utf8'  
);
```

- **schema_name** indicates the schema that the remote cluster table belongs to. If this option is not specified, **schema_name** is set to the schema of the foreign table.
- **table_name** indicates the remote cluster table name. If this option is not specified, **table_name** is set to the name of the foreign table.
- **encoding** indicates the encoding format of the remote cluster. If this option is not specified, the default encoding format of the source cluster database is used.

NOTE

- The values of **schema_name** and **table_name** are case sensitive and must be the same as those of the remote schema and table.
- The foreign table for interconnection cannot contain any constraints in its columns.
- The column names and column types of the foreign table must be the same as those of the **tbl_remote** table.
- **SERVER** must be set to the server created in [Step 1](#) and must contain the **syncsrv** attribute.

Step 3 Use the foreign table for interconnection to synchronize data.

- If the local cluster is the destination cluster, you can run the following statements:
Full data synchronization of all columns:

```
INSERT INTO tbl_local SELECT * FROM ft_tbl;
```


Data synchronization of all columns based on filter criteria:

```
INSERT INTO tbl_local SELECT * FROM ft_tbl WHERE col_2 = XX;
```


Full data synchronization of some columns:

```
INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl;
```


Data synchronization of some columns based on filter criteria:

```
INSERT INTO tbl_local (col_1) SELECT col_1 FROM ft_tbl WHERE col_2 = XX;
```
- If the local cluster is the source cluster, you can run the following statements:
Synchronization of unsharded tables:

```
INSERT INTO ft_tbl SELECT * FROM tbl_local;
```


Data synchronization of the **join** results:

```
INSERT INTO ft_tbl SELECT * FROM tbl_local1 join tbl_local2 ON XXX;
```

NOTE

- If a connection failure is reported, check the server information and ensure that both clusters are connected.
- If an error indicating GDS connection failure is reported, check whether the GDS server specified by **syncsrv** has been started and whether it can communicate with all nodes in both clusters.
- If an error is reported indicating that the table does not exist, check whether the **option** information of the foreign table is correct.
- If an error is reported indicating that the column does not exist, check whether the column name of the foreign table is the same as that of the source table.
- If an error message is displayed indicating that a column is repeatedly defined, check whether the column name is too long. If yes, use the AS alias to simplify the column name.
- If an error is reported indicating that the column type cannot be parsed, check whether the statement contains a column expression.
- If a column mismatch error is reported, check whether the column information of the foreign table is the same as that of the corresponding table in the remote cluster.
- If an error is reported indicating that the syntax is not supported, check whether complex syntax is used, such as **join**, **distinct**, and **sort**.
- If garbled characters are displayed, check whether the encoding formats of both databases are the same.
- If the local cluster is the source cluster, there is a low probability that data is successfully synchronized to the remote cluster but the local cluster returns an execution failure. In this case, you are advised to check the number of synchronized data records.
- If the local cluster is the source cluster, data synchronization controlled by transaction blocks and sub-transactions can be queried only after the total transaction is committed.

Step 4 Delete the foreign table for interconnection.

```
DROP FOREIGN TABLE ft_tbl;
```

----End

2.3.5 Using a gsql Meta-Command to Import Data

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

\copy Command

For details about the **\copy** command, see [Table 2-18](#).

Table 2-18 \copy meta-command

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

Parameter Description

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

- **binary**
Specifies that data is stored and read in binary mode instead of text mode. In binary mode, you cannot declare **DELIMITER**, **NULL**, or **CSV**. After specifying **BINARY**, **CSV**, **FIXED** and **TEXT** cannot be specified through **option** or **copy_option**.
- **oid**
Specifies the internal OID to be copied for each row.

 NOTE

An error is raised if OIDs are specified for a table that does not have OIDs, or in the case of copying a query.

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

Default value: **false/off**

- **delimiter [as] 'character'**
Specifies the character that separates columns within each row (line) of the file.

 NOTE

- A delimiter cannot be **\r** or **\n**.
- A delimiter cannot be the same as the **null** value. The delimiter of CSV data cannot be same as the **quote** value.
- The delimiter of TEXT data cannot contain any of the following characters:
`\.abcdefghijklmnopqrstuvwxyz0123456789`
- The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
- You are advised to use multi-characters and invisible characters for delimiters. For example, you can use the multiple-character delimiter "\$^&" and invisible delimiters, such as `E'\x07'`, `E'\x08'`, and `E'\x1b'`.

Value range: a multi-character delimiter within 10 bytes.

Default value:

- A tab character in TEXT format
- A comma (,) in CSV format
- No delimiter in FIXED format

- **null [as] 'string'**
Specifies that a string represents a null value in a data file.

Value range:

- A null value cannot be **\r** or **\n**. The maximum length is 100 characters.
- A null value cannot be the same as the **delimiter** or **quote** value.

Default value:

- An empty string without quotation marks in CSV format
- **\N** in TEXT format

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

In data import scenarios, if **header** is **on**, the first row of the data file will be identified as the header and ignored. If **header** is **off**, the first row will be identified as a data row.

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

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

Default value: **false/off**

- **quote** [as] 'character'

Specifies the quote character used when a data value is referenced in a CSV file.

Default value: double quotation mark (").

NOTE

- The **quote** value cannot be the same as the **delimiter** or **null** value.
- The **quote** value must be a single-byte character.
- You are advised to use invisible characters as quotes, for example, E'\x07', E'\x08', and E'\x1b'.
- **escape** [as] 'character'
This option is allowed only when using CSV format. This must be a single one-byte character.
Default value: double quotation mark ("). If the value is the same as the **quote** value, it will be replaced with \0.
- **force quote column_list** | *
Quotes all not-null values in each declared column when **CSV COPY TO** is used. NULL values will not be quoted.
Value range: an existing column.
- **force not null column_list**
In CSV COPY FROM mode, processes each specified column as though it were quoted and hence not a null value.
Value range: an existing column.

Example

Create the target table **copy_example**.

```
create table copy_example
(
  col_1 integer,
  col_2 text,
  col_3 varchar(12),
  col_4 date,
  col_5 time
);
```

- Example 1: Copy data from **stdin** to the target table **copy_example**.
`\copy copy_example from stdin csv;`

When you see the >> characters, you can start entering data. To finish your input, type a backslash and a period (\.).

Enter data to be copied followed by a newline.

End with a backslash and a period on a line by itself.

```
>> 1,"iamtext","iamvarchar",2006-07-07,12:00:00
>> \.
```

- Example 2: The **example.csv** file is in the local directory **/local/data/** and the file contains the header line. **()** is used as the delimiter, and the double quotation marks are used for **quote**. The content is as follows:

iamheader

1|"iamtext"|"iamvarchar"|2006-07-07|12:00:00

2|"iamtext"|"iamvarchar"|2022-07-07|19:00:02

Import data from the local file **example.csv** to the target table **copy_example**. If the header option is **on**, the first row is automatically ignored. By default, quotation marks are used for **quote**.

```
\copy copy_example from '/local/data/example.csv' with(header 'on', format 'csv', delimiter '|', date_format 'yyyy-mm-dd', time_format 'hh24:mi:ss');
```

- Example 3: The **example.csv** file is in the local directory **/local/data/**. The comma (,) is used as the delimiter, and the quotation mark (") is used for **quote**. The last field is missing in the first line, and one extra field is added in the second line. The content is as follows:

1,"iamtext","iamvarchar",2006-07-07

2,"iamtext","iamvarchar",2022-07-07,19:00:02,12:00:00

To import data from the local file **example.csv** to the target table **copy_example**, you don't need to specify the delimiter since the default delimiter is (,). Additionally, the fault tolerance parameters **IGNORE_EXTRA_DATA** and **FILL_MISSING_FIELDS** are set, which means that missing fields will be replaced with **NULL** and extra fields will be ignored.

```
\copy copy_example from '/local/data/example.csv' with( format 'csv', date_format 'yyyy-mm-dd', time_format 'hh24:mi:ss', IGNORE_EXTRA_DATA 'true', FILL_MISSING_FIELDS 'true');
```

- Example 4: Export the content of the **copy_example** table to **stdout** in CSV format, use double quotation marks as for **quote**, and use quotes to enclose the fourth and fifth columns.

```
\copy copy_example to stdout CSV quote as '"' force quote col_4,col_5;
```

2.3.6 Using COPY FROM STDIN to Import Data

2.3.6.1 Data Import Using COPY FROM STDIN

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

Use either of the following methods to write data to DWS using the **COPY FROM STDIN** statement:

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

2.3.6.2 Introduction to the CopyManager Class

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

Inheritance Relationship of CopyManager

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

```
public class CopyManager  
extends Object
```

Constructor Method

```
public CopyManager(BaseConnection connection)
```

throws SQLException

Basic Methods

Table 2-19 Basic methods of CopyManager

Return Value	Method	Description	Throws
CopyIn	copyIn(String sql)	-	SQLException
long	copyIn(String sql, InputStream from)	Uses COPY FROM STDIN to quickly import data to tables in a database from InputStream.	SQLException, IOException
long	copyIn(String sql, InputStream from, int bufferSize)	Uses COPY FROM STDIN to quickly import data to tables in a database from InputStream.	SQLException, IOException
long	copyIn(String sql, Reader from)	Uses COPY FROM STDIN to quickly import data to tables in a database from Reader.	SQLException, IOException
long	copyIn(String sql, Reader from, int bufferSize)	Uses COPY FROM STDIN to quickly import data to tables in a database from Reader.	SQLException, IOException
CopyOut	copyOut(String sql)	-	SQLException

Return Value	Method	Description	Throws
long	copyOut(String sql, OutputStream to)	Sends the result set of COPY TO STDOUT from a database to the OutputStream class.	SQLException,IOException
long	copyOut(String sql, Writer to)	Sends the result set of COPY TO STDOUT from a database to the Writer class.	SQLException,IOException

2.4 Full Database Migration

2.4.1 Using CDM to Migrate Data to DWS

You can use CDM to migrate data from other data sources (for example, MySQL) to the databases in DWS clusters.

CDM can migrate various types of data in batches between homogeneous and heterogeneous data sources.

NOTE

CDM migrates data to DWS using the copy method or the GDS parallel import method.

To learn more about how CDM is used to migrate data to DWS, see "DataArts Migration" in the *DataArts Studio User Guide*.

Figure 2-6 Migrating data to DWS using CDM



2.4.2 Using DSC to Migrate SQL Scripts

The DSC is a CLI tool running on the Linux or Windows OS. It is dedicated to providing customers with simple, fast, and reliable application SQL script migration services. It parses the SQL scripts of source database applications using the built-in syntax migration logic, and converts them to SQL scripts applicable to DWS databases. You do not need to connect the DSC to a database. It can migrate data in offline mode without service interruption. In DWS, you can run the migrated SQL scripts to restore the database, thereby easily migrating offline databases to the cloud.

The DSC can migrate SQL scripts of Teradata, Oracle, Netezza, MySQL, and DB2 databases.

Downloading the DSC SQL Migration Tool

Step 1 Log in to the DWS console.

Step 2 In the navigation tree on the left, choose **Management > Client Connections**.

Step 3 In the **Download Client and Driver** area, click **here** to download the DSC migration tool.

If you have clusters of different versions, the system displays a dialog box, prompting you to select the cluster version and download the client corresponding to the cluster version. In the cluster list on the **Cluster Management** page, click the name of the specified cluster and click the **Basic Information** tab to view the cluster version.

Step 4 After downloading the DSC tool to the local PC, use WinSCP to upload it to a Linux host.

The user who uploads the tool must have the full control permission on the target directory of the Linux host.

----End

Operation Guide for the DSC SQL Syntax Migration Tool

For details, see "DSC - SQL Syntax Migration Tool" in the *Data Warehouse Service Tool Guide*.

2.5 Metadata Migration

2.5.1 Using gs_dump and gs_dumpall to Export Metadata

2.5.1.1 Overview

DWS provides `gs_dump` and `gs_dumpall` to export required database objects and related information. To migrate database information, you can use a tool to import the exported metadata to a target database. `gs_dump` exports a single database or its objects. `gs_dumpall` exports all databases or global objects in a cluster. For details, see [Table 2-20](#).

Table 2-20 Application scenarios

Applica tion Scenari o	Export Granularity	Export Format	Import Method
Exportin g a single databas e	Database-level export - Export full information of a database. You can use the exported information to create a same database containing the same data as the current one. - Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same database as the current one, without data. - Export data of a database.	- Plain text - Custom - Directory .tar	- For details about how to import data files in text format, see Using a gsutil Meta-Command to Import Data. - For details about how to import data files in .tar, directory, or custom format, see Using gs_restore to Import Data.
	Schema-level export - Export full information of a schema. - Export data of a schema. - Export all object definitions of a schema, including the definitions of tables, stored procedures, and indexes.		
	Table-level export - Export full information of a table. - Export data of a table. - Export the definition of a table.		

Applica tion Scenari o	Export Granularity	Export Format	Import Method
Exportin g all databas es in a cluster	Database-level export - Export full information of a cluster. You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one. - Export all object definitions of a cluster, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures. You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data. - Export data of a cluster.	Plain text	For details about how to import data files, see Using a gsqr Meta-Command to Import Data .
	Global object export - Export tablespaces. - Export roles. - Export tablespaces and roles.		

gs_dump and gs_dumpall use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the export command to the role that has the permission. Then, gs_dump or gs_dumpall uses the specified role to export data. See [Table 2-20](#) for application scenarios and [Data Export By a User Without Required Permissions](#) for operation details.

gs_dump and gs_dumpall encrypt the exported data files. These files are decrypted before being imported to prevent data disclosure for higher database security.

When gs_dump and gs_dumpall are used to export data, the exported tables are locked, which blocks read and write operations.

gs_dump and gs_dumpall can export complete, consistent data. For example, if gs_dump is used to export database A or gs_dumpall is used to export all databases from a cluster at T1, data of database A or all databases in the cluster at that time point will be exported, and modifications on the databases after that time point will not be exported.

Obtain `gs_dump` and `gs_dumpall` by decompressing the **gsqL CLI client** package.

Precautions

- Do not modify an exported file or its content. Otherwise, restoration may fail.
- For data consistency and integrity, `gs_dump` and `gs_dumpall` set a share lock for a table to be dumped. If a share lock has been set for the table in other transactions, `gs_dump` and `gs_dumpall` lock the table after the lock is released. If the table cannot be locked within the specified time, the dump fails. You can customize the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.
- During an export, `gs_dumpall` reads all tables in a database. Therefore, you need to connect to the database as a cluster administrator to export a complete file. When you use `gsqL` to import scripts, cluster administrator permissions are also required to add users and user groups, and create databases.
- By default, the definitions of all views in the DWS database contain the prefix of table names or aliases (in **tab.col** format). Therefore, the definitions may be inconsistent with the original ones. As a result, the base table corresponding to the rebuilt view column is incorrect and an error is reported. However, this rarely happens. To prevent this problem, you are advised to set the GUC parameter **behavior_compat_options** to **compat_display_ref_table** when exporting view definitions, so the exported definitions are consistent with the original statements.

2.5.1.2 Exporting a Single Database

2.5.1.2.1 Exporting a Database

You can use `gs_dump` to export data and all object definitions of a database from DWS. You can specify the information to be exported as follows:

- Export full information of a database, including its data and all object definitions.
You can use the exported information to create a same database containing the same data as the current one.
- Export all object definitions of a database, including the definitions of the database, functions, schemas, tables, indexes, and stored procedures.
You can use the exported object definitions to quickly create a same database as the current one, without data.
- Export data of a database.

Procedure

- Step 1** Prepare an ECS as the `gsqL` client host. For details, see "Preparing an ECS as the `gsqL` Client Host" in the Data Warehouse Service (DWS) User Guide.
- Step 2** Download the `gsqL` client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where `gsqL` is to be installed. For details, see "Downloading the Client" in *Data Warehouse Service User Guide*.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Step 3 Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>  
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

If the following information is displayed, the DWS client is successfully configured:

```
All things done.
```

Step 5 Use `gs_dump` to export data of the database **gaussdb**.

```
gs_dump -W password -U jack -f /home//backup/postgres_backup.tar -p 8000 gaussdb -h 10.10.10.100 -F t
```

Table 2-21 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W <i>Password</i>
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ <i>postgres_backup.tar</i>
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000

Parameter	Description	Example Value
-h	<i>Cluster address.</i> If a public network address is used for connection, set this parameter to Public Network Address or Public Network Domain Name . If a private network address is used for connection, set this parameter to Private Network Address or Private Network Domain Name .	-h 10.10.10.100
dbname	Name of the database to be exported.	gaussdb
-F	Format of exported files. The values of -F are as follows: • p: plain text • c: custom • d: directory • t: .tar	-F t

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use `gs_dump` to run the following command to export full information of the database **gaussdb** and compress the exported files in SQL format.

```
gs_dump -W password -U jack -f /home//backup/postgres_backup.sql -p 8000 -h 10.10.10.100 gaussdb -Z 8 -F p
gs_dump[port=""][gaussdb][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port=""][gaussdb][2017-07-21 15:36:13]: total time: 3793 ms
```

Example 2: Use `gs_dump` to run the following command to export data of the database **gaussdb**, excluding object definitions. The exported files are in a custom format.

```
gs_dump -W Password -U jack -f /home//backup/postgres_data_backup.dmp -p 8000 -h 10.10.10.100 gaussdb -a -F c
gs_dump[port=""][gaussdb][2017-07-21 15:36:13]: dump database gaussdb successfully
gs_dump[port=""][gaussdb][2017-07-21 15:36:13]: total time: 3793 ms
```

Example 3: Use `gs_dump` to run the following command to export object definitions of the database **gaussdb**. The exported files are in SQL format.

```
--Before the export, the nation table contains data.
select n_nationkey,n_name,n_regionkey from nation limit 3;
n_nationkey |      n_name      | n_regionkey
-----+-----+-----
0 | ALGERIA          | 0
3 | CANADA           | 1
11 | IRAQ             | 4
```


(3 rows)

```
gs_dump -W password -U jack -f /home//backup/postgres_def_backup.sql -p 8000 -h 10.10.10.100 gaussdb -s -F p
gs_dump[port=""][gaussdb][2017-07-20 15:04:14]: dump database gaussdb successfully
gs_dump[port=""][gaussdb][2017-07-20 15:04:14]: total time: 472 ms
```

Example 4: Use `gs_dump` to run the following command to export object definitions of the database **gaussdb**. The exported files are in text format and are encrypted.

```
gs_dump -W password -U jack -f /home//backup/postgres_def_backup.sql -p 8000 -h 10.10.10.100 gaussdb --with-encryption AES128 --with-key 1234567812345678 -s -F p
gs_dump[port=""][gaussdb][2018-11-14 11:25:18]: dump database gaussdb successfully
gs_dump[port=""][gaussdb][2018-11-14 11:25:18]: total time: 1161 ms
```

2.5.1.2.2 Exporting a Schema

You can use `gs_dump` to export data and all object definitions of a schema from DWS. You can export one or more specified schemas as needed. You can specify the information to be exported as follows:

- Export full information of a schema, including its data and object definitions.
- Export data of a schema, excluding its object definitions.
- Export the object definitions of a schema, including the definitions of tables, stored procedures, and indexes.

Procedure

- Step 1** Prepare an ECS as the `gsqL` client host. For details, see "Preparing an ECS as the `gsqL` Client Host" in the Data Warehouse Service (DWS) User Guide.
- Step 2** Download the `gsqL` client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where `gsqL` is to be installed. For details, see "Downloading the Client" in *Data Warehouse Service User Guide*.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

- Step 3** Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- `<Path_for_storing_the_client>`: Replace it with the actual path.
- `dws_client_8.1.x_redhat_x86.zip`: This is the client tool package of **RedHat x86**. Replace it with the actual one.

- Step 4** Run the following command to configure the DWS client:

```
source gsqL.env.sh
```

If the following information is displayed, the DWS client is successfully configured:

```
All things done.
```

- Step 5** Use `gs_dump` to run the following command to export the **hr** and **public** schemas.

```
gs_dump -W Password -U jack -f /home//backup/MPPDB_schema_backup -p 8000 -h 10.10.10.100 human_resource -n hr -F d
```

Table 2-22 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. - This parameter is not required for database administrators if the trust policy is used for authentication. - If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W <i>Password</i>
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ MPPDB_schema_backup
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	<i>Cluster address</i> . If a public network address is used for connection, set this parameter to Public Network Address or Public Network Domain Name . If a private network address is used for connection, set this parameter to Private Network Address or Private Network Domain Name .	-h 10.10.10.100
dbname	Name of the database to be exported.	human_resource
-n	Names of schemas to be exported. Data of the specified schemas will also be exported. - Single schema: Enter -n <i>schemaname</i>. - Multiple schemas: Enter -n <i>schemaname</i> for each schema.	- Single schema: -n hr - Multiple schemas: -n hr -n public

Parameter	Description	Example Value
-F	Format of exported files. The values of -F are as follows: • p: plain text • c: custom • d: directory • t: .tar	-F d

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use `gs_dump` to run the following command to export full information of the **hr** schema. The exported files are compressed and stored in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.sql -p 8000 -h 10.10.10.100
human_resource -n hr -Z 6 -F p
gs_dump[port=""][human_resource][2017-07-21 16:05:55]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 16:05:55]: total time: 2425 ms
```

Example 2: Use `gs_dump` to run the following command to export data of the **hr** schema. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_data_backup.tar -p 8000 -h 10.10.10.100
human_resource -n hr -a -F t
gs_dump[port=""][human_resource][2018-11-14 15:07:16]: dump database human_resource successfully
gs_dump[port=""][human_resource][2018-11-14 15:07:16]: total time: 1865 ms
```

Example 3: Use `gs_dump` to run the following command to export the definition of the **hr** schema. The exported files are stored in a directory.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_def_backup -p 8000 -h 10.10.10.100
human_resource -n hr -s -F d
gs_dump[port=""][human_resource][2018-11-14 15:11:34]: dump database human_resource successfully
gs_dump[port=""][human_resource][2018-11-14 15:11:34]: total time: 1652 ms
```

Example 4: Use `gs_dump` to run the following command to export the **human_resource** database excluding the **hr** schema. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup.dmp -p 8000 -h 10.10.10.100
human_resource -N hr -F c
gs_dump[port=""][human_resource][2017-07-21 16:06:31]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 16:06:31]: total time: 2522 ms
```

Example 5: Use `gs_dump` to run the following command to export the object definitions of the **hr** and **public** schemas, encrypt the exported files, and store them in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup1.tar -p 8000 -h 10.10.10.100
human_resource -n hr -n public -s --with-encryption AES128 --with-key 1234567812345678 -F t
gs_dump[port=""][human_resource][2017-07-21 16:07:16]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 16:07:16]: total time: 2132 ms
```

Example 6: Use `gs_dump` to run the following command to export the **human_resource** database excluding the **hr** and **public** schemas. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_schema_backup2.dmp -p 8000 -h 10.10.10.100
human_resource -N hr -N public -F c
```

```
gs_dump[port=""][human_resource][2017-07-21 16:07:55]: dump database human_resource successfully  
gs_dump[port=""][human_resource][2017-07-21 16:07:55]: total time: 2296 ms
```

Example 7: Use `gs_dump` to run the following command to export all tables, including views, sequences, and foreign tables, in the **public** schema, and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_backup3.dmp -p 8000 -h 10.10.10.100  
human_resource -t public.* -t hr.staffs -F c  
gs_dump[port=""][human_resource][2018-12-13 09:40:24]: dump database human_resource successfully  
gs_dump[port=""][human_resource][2018-12-13 09:40:24]: total time: 896 ms
```

2.5.1.2.3 Exporting a Table

You can use `gs_dump` to export data and all object definitions of a table-level object from DWS. Views, sequences, and foreign tables are special tables. You can export one or more specified tables as needed. You can specify the information to be exported as follows:

- Export full information of a table, including its data and definition.
- Export data of a table.
- Export the definition of a table.

Procedure

- Step 1** Prepare an ECS as the `gsql` client host. For details, see "Preparing an ECS as the `gsql` Client Host" in the Data Warehouse Service (DWS) User Guide.
- Step 2** Download the `gsql` client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where `gsql` is to be installed. For details, see "Downloading the Client" in *Data Warehouse Service User Guide*.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

- Step 3** Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>  
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- `<Path_for_storing_the_client>`: Replace it with the actual path.
- `dws_client_8.1.x_redhat_x86.zip`: This is the client tool package of **RedHat x86**. Replace it with the actual one.

- Step 4** Run the following command to configure the DWS client:

```
source gsql_env.sh
```

If the following information is displayed, the DWS client is successfully configured:

```
All things done.
```

- Step 5** Use `gs_dump` to run the following command to export the **hr.staffs** and **hr employments** tables.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup -p 8000 -h 10.10.10.100  
human_resource -t hr.staffs -F d
```

Table 2-23 Common parameters

Parameter	Description	Example Value
-U	Username for connecting to the database. If this parameter is not configured, the username of the connected database is used.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W <i>password</i>
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/MPPDB_table_backup
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	<i>Cluster address</i> . If a public network address is used for connection, set this parameter to Public Network Address or Public Network Domain Name . If a private network address is used for connection, set this parameter to Private Network Address or Private Network Domain Name .	-h 10.10.10.100
dbname	Name of the database to be exported.	human_resource

Parameter	Description	Example Value
-t	Table (or view, sequence, foreign table) to be exported. You can specify multiple tables by listing them or using wildcard characters. When you use wildcard characters, quote wildcard patterns with single quotation marks (") to prevent the shell from expanding the wildcard characters. - Single table: Enter -t <i>schema.table</i>. - Multiple tables: Enter -t <i>schema.table</i> for each table.	- Single table: -t <i>hr.staffs</i> - Multiple tables: -t <i>hr.staffs</i> -t <i>hr employments</i>
-F	Format of exported files. The values of -F are as follows: p: plain text c: custom d: directory t: .tar	-F d

For details about other parameters, see "gs_dump" in the *Tool Guide*.

----End

Examples

Example 1: Use `gs_dump` to run the following command to export full information of the **hr.staffs** table. The exported files are in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -Z 6 -F p
gs_dump[port=""][human_resource][2017-07-21 17:05:10]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 17:05:10]: total time: 3116 ms
```

Example 2: Use `gs_dump` to run the following command to export data of the **hr.staffs** table. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_data_backup.tar -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -a -F t
gs_dump[port=""][human_resource][2017-07-21 17:04:26]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 17:04:26]: total time: 2570 ms
```

Example 3: Use `gs_dump` to run the following command to export the definition of the **hr.staffs** table. The exported files are stored in a directory.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_def_backup -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -s -F d
gs_dump[port=""][human_resource][2017-07-21 17:03:09]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 17:03:09]: total time: 2297 ms
```

Example 4: Use `gs_dump` to run the following command to export the **human_resource** database excluding the **hr.staffs** table. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup4.dmp -p 8000 -h 10.10.10.100
human_resource -T hr.staffs -F c
gs_dump[port=""][human_resource][2017-07-21 17:14:11]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 17:14:11]: total time: 2450 ms
```

Example 5: Use `gs_dump` to run the following command to export the **hr.staffs** and **hr employments** tables. The exported files are in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup1.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -t hr.employments -F p
gs_dump[port=""][human_resource][2017-07-21 17:19:42]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 17:19:42]: total time: 2414 ms
```

Example 6: Use `gs_dump` to run the following command to export the **human_resource** database excluding the **hr.staffs** and **hr.employments** tables. The exported files are in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup2.sql -p 8000 -h 10.10.10.100
human_resource -T hr.staffs -T hr.employments -F p
gs_dump[port=""][human_resource][2017-07-21 17:21:02]: dump database human_resource successfully
gs_dump[port=""][human_resource][2017-07-21 17:21:02]: total time: 3165 ms
```

Example 7: Use `gs_dump` to run the following command to export data and definition of the **hr.staffs** table, and the definition of the **hr.employments** table. The exported files are in .tar format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup3.tar -p 8000 -h 10.10.10.100
human_resource -t hr.staffs -t hr.employments --exclude-table-data hr.employments -F t
gs_dump[port=""][human_resource][2018-11-14 11:32:02]: dump database human_resource successfully
gs_dump[port=""][human_resource][2018-11-14 11:32:02]: total time: 1645 ms
```

Example 8: Use `gs_dump` to run the following command to export data and definition of the **hr.staffs** table, encrypt the exported files, and store them in text format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup4.sql -p 8000 -h 10.10.10.100
human_resource -t hr.staffs --with-encryption AES128 --with-key 1212121212121212 -F p
gs_dump[port=""][human_resource][2018-11-14 11:35:30]: dump database human_resource successfully
gs_dump[port=""][human_resource][2018-11-14 11:35:30]: total time: 6708 ms
```

Example 9: Use `gs_dump` to run the following command to export all tables, including views, sequences, and foreign tables, in the **public** schema, and the **staffs** table in the **hr** schema, including data and table definition. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_table_backup5.dmp -p 8000 -h 10.10.10.100
human_resource -t public.* -t hr.staffs -F c
gs_dump[port=""][human_resource][2018-12-13 09:40:24]: dump database human_resource successfully
gs_dump[port=""][human_resource][2018-12-13 09:40:24]: total time: 896 ms
```

Example 10: Use `gs_dump` to run the following command to export the definition of the view referencing to the **test1** table in the **t1** schema. The exported files are in a custom format.

```
gs_dump -W password -U jack -f /home//backup/MPPDB_view_backup6 -p 8000 -h 10.10.10.100
human_resource -t t1.test1 --include-depend-objs --exclude-self -F d
gs_dump[port=""][jack][2018-11-14 17:21:18]: dump database human_resource successfully
gs_dump[port=""][jack][2018-11-14 17:21:23]: total time: 4239 ms
```

2.5.1.3 Exporting All Databases

2.5.1.3.1 Exporting All Databases

You can use `gs_dumpall` to export full information of all databases in a cluster from DWS, including information about each database and global objects in the cluster. You can specify the information to be exported as follows:

- Export full information of all databases, including information about each database and global objects (such as roles and tablespaces) in the cluster.
You can use the exported information to create a same cluster containing the same databases, global objects, and data as the current one.
- Export data of all databases, excluding all object definitions and global objects.
- Export all object definitions of all databases, including the definitions of tablespaces, databases, functions, schemas, tables, indexes, and stored procedures.

You can use the exported object definitions to quickly create a same cluster as the current one, containing the same databases and tablespaces but without data.

Procedure

- Step 1** Prepare an ECS as the `gsq` client host. For details, see "Preparing an ECS as the `gsq` Client Host" in the Data Warehouse Service (DWS) User Guide.
- Step 2** Download the `gsq` client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where `gsq` is to be installed. For details, see "Downloading the Client" in *Data Warehouse Service User Guide*.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

- Step 3** Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- `<Path_for_storing_the_client>`: Replace it with the actual path.
- `dws_client_8.1.x_redhat_x86.zip`: This is the client tool package of **RedHat x86**. Replace it with the actual one.

- Step 4** Run the following command to configure the DWS client:

```
source gsql_env.sh
```

If the following information is displayed, the DWS client is successfully configured:

```
All things done.
```

- Step 5** Use `gs_dumpall` to run the following command to export information of all databases.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100
```


Table 2-24 Common parameters

Parameter	Description	Example Value
-U	Username for database connection. The user must be a cluster administrator.	-U dbadmin
-W	User password for database connection. - This parameter is not required for database administrators if the trust policy is used for authentication. - If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W <i>Password</i>
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home/dbadmin/backup/ MPPDB_backup.sql
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	<i>Cluster address</i> . If a public network address is used for connection, set this parameter to Public Network Address or Public Network Domain Name . If a private network address is used for connection, set this parameter to Private Network Address or Private Network Domain Name .	-h 10.10.10.100

For details about other parameters, see "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of all databases in a cluster. The exported files are in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of

the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100
gs_dumpall[port=""][2017-07-21 15:57:31]: dumpall operation successful
gs_dumpall[port=""][2017-07-21 15:57:31]: total time: 9627 ms
```

Example 2: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export definitions of all databases in a cluster. The exported files are in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100 -s
gs_dumpall[port=""][2018-11-14 11:28:14]: dumpall operation successful
gs_dumpall[port=""][2018-11-14 11:28:14]: total time: 4147 ms
```

Example 3: Use **gs_dumpall** to run the following command export data of all databases in a cluster, encrypt the exported files, and store them in text format. After the command is executed, a large amount of output information will be displayed. **total time** will be displayed at the end of the information, indicating that the export is successful. In this example, only related output information is included.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100 -a --with-encryption AES128 --with-key 1234567812345678
gs_dumpall[port=""][2018-11-14 11:32:26]: dumpall operation successful
gs_dumpall[port=""][2018-11-14 11:23:26]: total time: 4147 ms
```

2.5.1.3.2 Exporting Global Objects

You can use **gs_dumpall** to export global objects from DWS, including database users, user groups, tablespaces, and attributes (for example, global access permissions).

Procedure

- Step 1** Prepare an ECS as the **gsq** client host. For details, see "Preparing an ECS as the **gsq** Client Host" in the Data Warehouse Service (DWS) User Guide.
- Step 2** Download the **gsq** client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where **gsq** is to be installed. For details, see "Downloading the Client" in *Data Warehouse Service User Guide*.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

- Step 3** Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- **<Path_for_storing_the_client>**: Replace it with the actual path.
- **dws_client_8.1.x_redhat_x86.zip**: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

If the following information is displayed, the DWS client is successfully configured:

```
All things done.
```

Step 5 Use `gs_dumpall` to run the following command to export tablespace objects.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -h 10.10.10.100 -t
```

Table 2-25 Common parameters

Parameter	Description	Example Value
-U	Username for database connection. The user must be a cluster administrator.	-U dbadmin
-W	User password for database connection. - This parameter is not required for database administrators if the trust policy is used for authentication. - If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W <i>Password</i>
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/ <i>MPPDB_tablespace.sql</i>
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	<i>Cluster address</i> . If a public network address is used for connection, set this parameter to Public Network Address or Public Network Domain Name . If a private network address is used for connection, set this parameter to Private Network Address or Private Network Domain Name .	-h 10.10.10.100

Parameter	Description	Example Value
-t	Dumps only tablespaces. You can also use --tablespaces-only alternatively.	-

For details about other parameters, see "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of global tablespaces and users in a cluster. The exported files are in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_globals.sql -p 8000 -h 10.10.10.100 -g
gs_dumpall[port=""][2018-11-14 19:06:24]: dumpall operation successful
gs_dumpall[port=""][2018-11-14 19:06:24]: total time: 1150 ms
```

Example 2: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export global tablespaces in a cluster, encrypt the exported files, and store them in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_tablespace.sql -p 8000 -h 10.10.10.100 -t --with-encryption AES128 --with-key 1212121212121212
gs_dumpall[port=""][2018-11-14 19:00:58]: dumpall operation successful
gs_dumpall[port=""][2018-11-14 19:00:58]: total time: 186 ms
```

Example 3: Use **gs_dumpall** to run the following command as the cluster administrator **dbadmin** to export information of global users in a cluster. The exported files are in text format.

```
gs_dumpall -W password -U dbadmin -f /home/dbadmin/backup/MPPDB_user.sql -p 8000 -h 10.10.10.100 -r
gs_dumpall[port=""][2018-11-14 19:03:18]: dumpall operation successful
gs_dumpall[port=""][2018-11-14 19:03:18]: total time: 162 ms
```

2.5.1.4 Data Export By a User Without Required Permissions

gs_dump and **gs_dumpall** use **-U** to specify the user that performs the export. If the specified user does not have the required permission, data cannot be exported. In this case, you can set **--role** in the export command to the role that has the permission. Then, **gs_dump** or **gs_dumpall** uses the specified role to export data.

Procedure

- Step 1** Prepare an ECS as the gsql client host. For details, see "Preparing an ECS as the gsql Client Host" in the Data Warehouse Service (DWS) User Guide.
- Step 2** Download the gsql client and use an SSH transfer tool (such as WinSCP) to upload it to the Linux server where gsql is to be installed. For details, see "Downloading the Client" in *Data Warehouse Service User Guide*.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Step 3 Run the following commands to decompress the client:

```
cd <Path_for_storing_the_client>
unzip dws_client_8.x.x_redhat_x64.zip
```

Where,

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x86.zip*: This is the client tool package of **RedHat x86**. Replace it with the actual one.

Step 4 Run the following command to configure the DWS client:

```
source gsql_env.sh
```

If the following information is displayed, the DWS client is successfully configured:

```
All things done.
```

Step 5 Use `gs_dump` to export data of the **human_resource** database.

User **jack** does not have the permission for exporting data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set `--role` to **role1** in the export command. The exported files are in .tar format.

```
gs_dump -U jack -W password -f /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100
human_resource --role role1 --rolepassword password -F t
```

Table 2-26 Common parameters

Parameter	Description	Example Value (dbadmin)
-U	Username for database connection.	-U jack
-W	User password for database connection. • This parameter is not required for database administrators if the trust policy is used for authentication. • If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W Password
-f	Folder to store exported files. If this parameter is not specified, the exported files are stored in the standard output.	-f /home//backup/MPPDB_backup.tar

Parameter	Description	Example Value (dbadmin)
-p	Name extension of the TCP port on which the server is listening or the local Unix domain socket. This parameter is configured to ensure connections.	-p 8000
-h	<i>Cluster address</i> : If a public network address is used for connection, set this parameter to Public Network Address or Public Network Domain Name . If a private network address is used for connection, set this parameter to Private Network Address or Private Network Domain Name .	-h 10.10.10.100
dbname	Name of the database to be exported.	human_resource
--role	Role name for the export operation. After this parameter is set and gs_dump or gs_dumpall connects to the database, the SET ROLE command will be issued. When the user specified by -U does not have the permissions required by gs_dump or gs_dumpall, this parameter allows the user to switch to a role with the required permissions.	-r role1
--rolepassword	Role password.	--rolepassword password
-F	Format of exported files. The values of -F are as follows: • p: plain text • c: custom • d: directory • t: .tar	-F t

For details about other parameters, see "gs_dump" or "gs_dumpall" in the *Tool Guide*.

----End

Examples

Example 1: User **jack** does not have the permission for exporting data of the **human_resource** database and the role **role1** has this permission. To export data of the **human_resource** database, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup11.tar -p 8000 -h 10.10.10.100
human_resource --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 4239 ms
```

Example 2: User **jack** does not have the permission for exporting the **public** schema and the role **role1** has this permission. To export the **public** schema, you can set **--role** to **role1** in the export command. The exported files are in .tar format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "1234@abc";

gs_dump -U jack -W password -f /home//backup/MPPDB_backup12.tar -p 8000 -h 10.10.10.100
human_resource -n public --role role1 --rolepassword password -F t
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: dump database human_resource successfully
gs_dump[port='8000'][human_resource][2017-07-21 16:21:10]: total time: 3278 ms
```

Example 3: User **jack** does not have the permission for exporting all databases in a cluster and the role **role1** has this permission. To export all databases, you can set **--role** to **role1** in the export command. The exported files are in text format.

```
human_resource=# CREATE USER jack IDENTIFIED BY "password";

gs_dumpall -U jack -W password -f /home//backup/MPPDB_backup.sql -p 8000 -h 10.10.10.100 --role role1
--rolepassword password
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: dumpall operation successful
gs_dumpall[port='8000'][human_resource][2018-11-14 17:26:18]: total time: 6437 ms
```

2.5.2 Using gs_restore to Import Data

Scenarios

gs_restore is an import tool provided by DWS. You can use **gs_restore** to import the files exported by **gs_dump** to a database. **gs_restore** can import the files in .tar, custom, or directory format.

gs_restore can:

- Import data to a database.
If a database is specified, data is imported to the database. If multiple databases are specified, the password for connecting to each database also needs to be specified.
- Import data to a script.
If no database is specified, a script containing the SQL statement to recreate the database is created and written to a file or standard output. This script output is equivalent to the plain text output of **gs_dump**.

You can specify and sort the data to be imported.

Procedure

NOTE

gs_restore incrementally imports data by default. To avoid data exceptions caused by consecutive imports, use the **-e** and **-c** parameters for each import. This will delete existing data from the target database before each import and exit the import task with an error message (which is displayed after the import process is complete) before proceeding with the next import.

Step 1 Log in to the server as the **root** user and run the following command to go to the data storage path:

```
cd /opt/bin
```

Step 2 Use **gs_restore** to import all object definitions from the exported file of the whole **postgres** database to the **backupdb** database.

```
gs_restore -W password -U jack /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb -s -e -c
```

Table 2-27 Common parameters

Parameters	Description	Example Value
-U	Username for database connection.	-U jack
-W	User password for database connection. - This parameter is not required for database administrators if the trust policy is used for authentication. - If you connect to the database without specifying this parameter and you are not a database administrator, you will be prompted to enter the password.	-W <i>Password</i>
-d	Database to which data will be imported.	-d backupdb
-p	TCP port or the local Unix-domain socket file extension on which the server is listening for connections.	-p 8000

Parameters	Description	Example Value
-h	<i>Cluster address.</i> If a public network address is used for connection, set this parameter to Public Network Address or Public Network Domain Name . If a private network address is used for connection, set this parameter to Private Network Address or Private Network Domain Name .	-h 10.10.10.100
-e	Exits the current import task and performs the next if an error occurs when you send a SQL statement in the current import task. Error messages are displayed after the import process is complete.	-
-c	Cleans existing objects from the target database before the import.	-
-s	Imports only object definitions in schemas and does not import data. Sequence values will also not be imported.	-

For details about other parameters, see "Server Tools" > "gs_restore" in the *Data Warehouse Service (DWS) Tool Reference*.

----End

Examples

Example 1: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup.dmp** file (custom format).

```
gs_restore -W password backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore: total time: 13053 ms
```

Example 2: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup.tar** file.

```
gs_restore backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:21:32]: restore operation successful
gs_restore[2017-07-21 19:21:32]: total time: 21203 ms
```

Example 3: Run **gs_restore** to import data and all object definitions of the **postgres** database from the **MPPDB_backup** directory.

```
gs_restore backup/MPPDB_backup -p 8000 -h 10.10.10.100 -d backupdb
gs_restore[2017-07-21 19:26:46]: restore operation successful
gs_restore[2017-07-21 19:26:46]: total time: 21003 ms
```

Example 4: Run **gs_restore** to import the definitions of all objects in the **postgres** database to the **backupdb** database. Table data is not imported.

```
gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb -s -e -c
gs_restore[2017-07-21 19:46:27]: restore operation successful
gs_restore[2017-07-21 19:46:27]: total time: 32993 ms
```

Example 5: Run **gs_restore** to import data and all definitions in the **PUBLIC** schema from the **MPPDB_backup.dmp** file. Existing objects are deleted from the target database before the import. If an existing object references to an object in another schema, you need to manually delete the referenced object first.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n PUBLIC
gs_restore: [archiver (db)] Error while PROCESSING TOC:
gs_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1 gaussdba
gs_restore: [archiver (db)] could not execute query: ERROR: cannot drop table table1 because other objects
depend on it
DETAIL:  view t1.v1 depends on table table1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
Command was: DROP TABLE public.table1;
```

Manually delete the referenced object and create it again after the import is complete.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -n PUBLIC
gs_restore[2017-07-21 19:52:26]: restore operation successful
gs_restore[2017-07-21 19:52:26]: total time: 2203 ms
```

Example 6: Run **gs_restore** to import the definition of the **hr.staffs** table in the **PUBLIC** schema from the **MPPDB_backup.dmp** file. Before the import, the **hr.staffs** table does not exist.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -c -s -n PUBLIC -t hr.staffs
gs_restore[2017-07-21 19:56:29]: restore operation successful
gs_restore[2017-07-21 19:56:29]: total time: 21000 ms
```

Example 7: Run **gs_restore** to import data of the **hr.staffs** table in **PUBLIC** schema from the **MPPDB_backup.dmp** file. Before the import, the **hr.staffs** table is empty.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -h 10.10.10.100 -d backupdb -e -a -n PUBLIC -t hr.staffs
gs_restore[2017-07-21 20:12:32]: restore operation successful
gs_restore[2017-07-21 20:12:32]: total time: 20203 ms
```

Example 8: Run **gs_restore** to import the definition of the **hr.staffs** table. Before the import, the **hr.staffs** table already exists.

```
human_resource=# select * from hr.staffs;
 staff_id | first_name | last_name | email | phone_number | hire_date | employment_id |
 salary | commission_pct | manager_id | section_id
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
 200 | Jennifer | Whalen | JWHALEN | 515.123.4444 | 1987-09-17 00:00:00 | AD_ASST |
4400.00 | | 101 | 10
 201 | Michael | Hartstein | MHARTSTE | 515.123.5555 | 1996-02-17 00:00:00 | MK_MAN |
13000.00 | | 100 | 20

gsql -d human_resource -p 8000
gsql ((GaussDB x.x.x build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

human_resource=# drop table hr.staffs CASCADE;
NOTICE: drop cascades to view hr.staff_details_view
```

```
gs_restore -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d human_resource -n  
hr -t staffs -s -e  
restore operation successful  
total time: 904 ms
```

```
human_resource=# select * from hr.staffs;  
staff_id | first_name | last_name | email | phone_number | hire_date | employment_id | salary |  
commission_pct | manager_id | section_id  
-----+-----+-----+-----+-----+-----+-----+-----+-----  
+-----+-----+-----  
(0 rows)
```

Example 9: Run **gs_restore** to import data and definitions of the **staffs** and **areas** tables. Before the import, the **staffs** and **areas** tables do not exist.

```
human_resource=# \d  
List of relations  
Schema | Name | Type | Owner | Storage  
-----+-----+-----+-----+-----  
hr | employment_history | table | | {orientation=row,compression=no}  
hr | employments | table | | {orientation=row,compression=no}  
hr | places | table | | {orientation=row,compression=no}  
hr | sections | table | | {orientation=row,compression=no}  
hr | states | table | | {orientation=row,compression=no}  
(5 rows)
```

```
gs_restore -W password /home/mppdb/backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d  
human_resource -n hr -t staffs -n hr -t areas  
restore operation successful  
total time: 724 ms
```

```
human_resource=# \d  
List of relations  
Schema | Name | Type | Owner | Storage  
-----+-----+-----+-----+-----  
hr | areas | table | | {orientation=row,compression=no}  
hr | employment_history | table | | {orientation=row,compression=no}  
hr | employments | table | | {orientation=row,compression=no}  
hr | places | table | | {orientation=row,compression=no}  
hr | sections | table | | {orientation=row,compression=no}  
hr | staffs | table | | {orientation=row,compression=no}  
hr | states | table | | {orientation=row,compression=no}  
(7 rows)
```

```
human_resource=# select * from hr.areas;  
area_id | area_name  
-----+-----  
4 | Iron  
1 | Wood  
2 | Lake  
3 | Desert  
(4 rows)
```

Example 10: Run **gs_restore** to import data and all object definitions in the **hr** schema.

```
gs_restore -W password /home//backup/MPPDB_backup1.sql -p 8000 -h 10.10.10.100 -d backupdb -n hr -e -  
c  
restore operation successful  
total time: 702 ms
```

Example 11: Run **gs_restore** to import all object definitions in the **hr** and **hr1** schemas to the **backupdb** database.

```
gs_restore -W password /home//backup/MPPDB_backup2.dmp -p 8000 -h 10.10.10.100 -d backupdb -n hr -  
n hr1 -s  
restore operation successful  
total time: 665 ms
```

Example 12: Run **gs_restore** to decrypt the files exported from the **human_resource** database and import them to the **backupdb** database.

```
create database backupdb;

gs_restore /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb --with-
key=1234567812345678
restore operation successful
total time: 23472 ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB x.x.x build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id | area_name
-----+-----
      4 | Iron
      1 | Wood
      2 | Lake
      3 | Desert
(4 rows)
```

Example 13: **user 1** does not have the permission to import data from an exported file to the **backupdb** database and **role1** has this permission. To import the exported data to the **backupdb** database, you can set **--role** to **role1** in the **gs_restore** command.

```
human_resource=# CREATE USER user1 IDENTIFIED BY 'password';

gs_restore -U user1 -W password /home//backup/MPPDB_backup.tar -p 8000 -h 10.10.10.100 -d backupdb
--role role1 --rolepassword password
restore operation successful
total time: 554 ms

gsql -d backupdb -p 8000 -r
gsql ((GaussDB x.x.x build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138 release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

backupdb=# select * from hr.areas;
 area_id | area_name
-----+-----
      4 | Iron
      1 | Wood
      2 | Lake
      3 | Desert
(4 rows)
```

2.6 Exporting Data

2.6.1 Exporting Data to OBS

2.6.1.1 Parallel OBS Data Export

Overview

When you export data in parallel using OBS foreign tables, data files to be exported are specified based on the export mode and exported data format included in OBS foreign table settings. Data is exported in parallel through multiple DNs from DWS to the data files, which improves the overall data export performance.

- The CN only plans data export tasks and delivers the tasks to DNs for execution. In this case, the CN is released to process external requests.
- Every DN is involved in data export, and the computing capabilities and bandwidths of all the DNs are fully leveraged to export data.
- You can concurrently export data using multiple OBS services, but the bucket and object paths specified for the export tasks must be different and cannot be null.
- The OBS server connects to DWS cluster nodes. The export rate is affected by the network bandwidth.
- The text, CSV, ORC, and Parquet formats are supported. The size of data in a single row must be less than 1 GB.
- To ensure the correctness of data import or export, you need to import or export data from OBS in the same compatibility mode.
For example, data imported or exported in MySQL compatibility mode can be exported or imported only in MySQL compatibility mode.

Currently, data can be exported using either of the following methods:

- Method 1: You do not need to create a server. Use the default server to create a foreign table. Data in TXT or CSV format is supported. For details, see [Exporting CSV/TXT Data to OBS](#).
- Method 2: Create a server and use it to create a foreign table. ORC and Parquet data formats are supported. For details, see [Exporting ORC and Parquet Data to OBS](#). Only 9.1.0 and later versions support data in Parquet format.

NOTICE

- Ensure that no Chinese characters are contained in paths used for importing data to or exporting data from OBS.
- Data cannot be imported to or exported from OBS across regions. Ensure that OBS and the DWS cluster are in the same region.
- To ensure the correctness of data import or export, you need to import or export data from OBS in the same compatibility mode.

If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Related Concepts

- **Source data file:** a text, CSV, ORC, or Parquet file that stores data.
- **OBS:** a cloud storage service used to store unstructured data, such as documents, images, and videos. Data objects concurrently exported from DWS are stored on the OBS server.
- **Bucket:** a container storing objects on OBS.
 - Object storage is a flat storage mode. Layered file system structures are not needed because all objects in buckets are at the same logical layer.
 - In OBS, each bucket name must be unique and cannot be changed. A default access control list (ACL) is created with a bucket. Each item in the

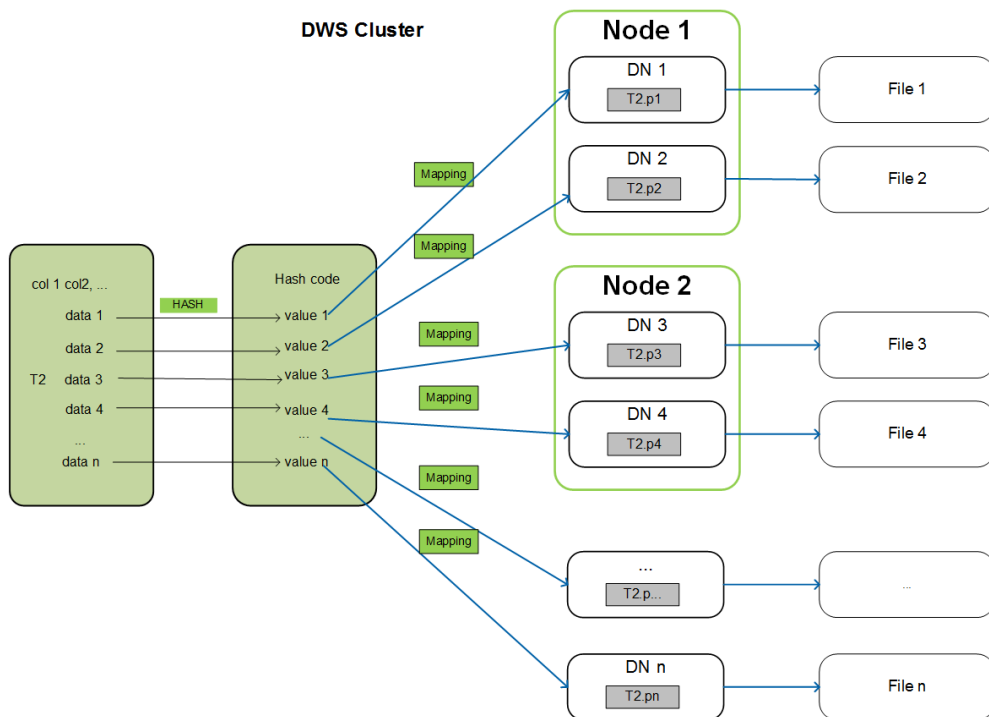
ACL contains permissions granted to certain users, such as **READ**, **WRITE**, and **FULL_CONTROL**. Only authorized users can perform bucket operations, such as creating, deleting, viewing, and setting ACLs for buckets.

- A user can create a maximum of 100 buckets. The total data size and the number of objects and files in each bucket are not limited.
- **Object**: a basic data storage unit in OBS. Data uploaded by users is stored in OBS buckets as objects. Object attributes include **Key**, **Metadata**, and **Data**.
Generally, objects are managed as files. However, OBS has no file system-related concepts, such as files or folders. To let users easily manage data, OBS allows them to simulate folders. Users can add a slash (/) in the object name, for example, **tpcds1000/stock.csv**. In this name, **tpcds1000** is regarded as the folder name and **stock.csv** the file name. The value of **key** (object name) is still **tpcds1000/stock.csv**, and the content of the object is the content of the **stock.csv** file.
- **Key**: name of an object. It is a UTF-8 character sequence containing 1 to 1024 characters. A key value must be unique in a bucket. Users can name the objects they stored or obtained as *Bucket name+Object name*.
- **Metadata**: object metadata, which contains information about the object. There are system metadata and user metadata. The metadata is uploaded to OBS as key-value pairs together with HTTP headers.
 - System metadata is generated by OBS and used for processing object data. System metadata includes **Date**, **Content-length**, **last-modify**, and **Content-MD5**.
 - User metadata contains object descriptions specified by users for uploading objects.
- **Data**: object content, which is regarded by OBS as stateless binary data.
- **Foreign table**: A foreign table is used to identify data in a source data file. It stores information, such as the location, format, destination location, encoding format, and data delimiter of a source data file.

Principles

The following describes the principles of exporting data from a cluster to OBS by using a distributed hash table or a replication table.

- **Distributed hash table**: the table for which **DISTRIBUTE BY HASH (Column_Name)** is specified in the table creation statement.
A distributed hash table stores data in hash mode. [Figure 2-7](#) shows how to export data from table (**T2**) to OBS as an example.
During table data storage, the **col2** hash column in table **T2** is hashed, and a hash value is generated. The tuple is distributed to corresponding DNs for storage according to the mapping between the DNs and the hash value.
When data is exported to OBS, DNs that store the exported data of **T2** directly export their data files to OBS. Original data on multiple nodes will be exported in parallel.

Figure 2-7 Hash distribution principle

- Replication table: the table for which **DISTRIBUTE BY REPLICATION** is specified in the table creation statement.
A replication table stores a package of complete table data on each DWS node. When exporting data to OBS, DWS randomly selects a DN for export.

Naming Rules of Exported Files

Rules for naming the files exported from DWS to OBS are as follows:

- Data exported from DNs is stored on OBS in segment format. The file is named as *Table name_Node name_segment.n*. *n* is a natural number starting from 0, for example, 0, 1, 2, 3.

For example, the data of table **t1** on **datanode3** will be exported as **t1_datanode3_segment.0**, **t1_datanode3_segment.1**, and so on.

You are advised to export data from different clusters or databases to different OBS buckets or different paths of the same OBS bucket.

- Each segment can store a maximum of 1 GB data, with no tuples sliced. If data stored in a segment exceeds 1 GB, the excess data will be stored in the second segment.

For example:

A segment has already stored 100 pieces of tuples (1023 MB) when **datanode3** exports data from **t1** to OBS. If a 5 MB tuple is inserted to the segment, the data size becomes 1028 MB. In this case, file **t1_datanode3_segment.0** (1023 MB) is generated and stored on OBS, and the new tuple is stored on OBS as file **t1_datanode3_segment.1**.

- When data is exported from a distributed hash table, the number of segments generated on each DN depends on the data volume stored on a DN, not on the number of DNs in the cluster. Data stored in hash mode may not be evenly distributed on each DN.

For example, a cluster has **DataNode1**, **DataNode2**, **DataNode3**, **DataNode4**, **DataNode5**, and **DataNode6**, which store 1.5 GB, 0.7 GB, 0.6 GB, 0.8 GB, 0.4 GB, and 0.5 GB data, respectively. Seven OBS segment files will be generated during data export because **DataNode1** will generate two segment files, which store 1 GB and 0.5 GB data, respectively.

Data Export Process

Figure 2-8 Concurrent data export

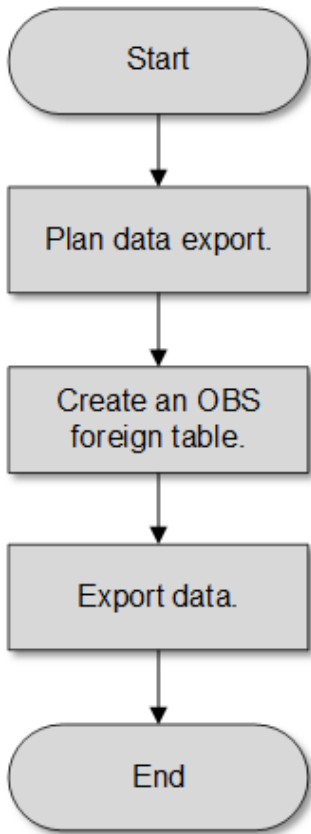


Table 2-28 Process description

Procedure	Description	Subtask
Plan data export.	Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data files. For details, see Planning Data Export .	-

Procedure	Description	Subtask
Create an OBS foreign table.	Create a foreign table to help OBS specify information about data files to be exported. The foreign table stores information, such as the destination location, format, encoding, and data delimiter of a source data file. For details, see Creating an OBS Foreign Table .	-
Export data.	After the foreign table is created, run the INSERT statement to efficiently export data to data files. For details, see Exporting Data .	-

2.6.1.2 Exporting CSV/TXT Data to OBS

2.6.1.2.1 Planning Data Export

Scenarios

Plan the storage location of exported data in OBS.

Planning OBS Save Path and File

You need to specify the OBS path (to directory) for storing data that you want to export. The exported data can be saved to a file in CSV format. The system also supports TEXT so that you can import the exported data to various applications.

The target directory cannot contain any files.

Planning OBS Bucket Permissions

The user used to export data must:

- Have OBS enabled.
- Have the write permission on the OBS bucket where the data export path is located.

You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

For details, see [Granting Write Permission to OBS Storage Location and OBS Bucket as Planned](#).

Planning Data to Be Exported and Foreign Tables

You must prepare data to be exported in the database table, and the data volume per row must be less than 1 GB. Based on the data to be exported, plan foreign tables whose attributes such as columns, column types, and length match those of user data.

Granting Write Permission to OBS Storage Location and OBS Bucket as Planned

Step 1 Create an OBS bucket and a folder in the OBS bucket as the directory for storing exported data.

1. Log in to OBS console.

Click **Service List** and choose **Object Storage Service** to open the OBS management console.

2. Create a bucket.

For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Buckets > Creating a Bucket" in the *Object Storage Service User Guide*.

For example, create two buckets named **mybucket** and **mybucket02**.

3. Create a folder.

For details about how to create an OBS bucket, see "OBS Console Operation Guide > Managing Objects > Creating a Folder" in the *Object Storage Service User Guide*.

Example:

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

Step 2 Determine the path of the created OBS folder.

Specify the OBS path for storing exported data files. This path is the value of the **location** parameter used for creating a foreign table.

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

In this example, the OBS folder path is as follows:

```
obs://mybucket/output_data/
```

NOTE

The OBS directory to be used for storing data files must be empty.

Step 3 Grant the OBS bucket write permission to the user who wants to export data.

When exporting data, a user must have the write permission on the OBS bucket where the data export path is located. You can configure ACL permissions for the OBS bucket to grant the write permission to a specific user.

For details, see "OBS Console Operation Guide > Permission Control > Configuring a Bucket ACL" in the *Object Storage Service User Guide*.

----End

2.6.1.2.2 Creating an OBS Foreign Table

Procedure

- Step 1** Based on the path planned in [Planning Data Export](#), determine the value of the **location** parameter used for creating a foreign table.
- Step 2** Obtain the access keys (AK and SK) to access OBS.
- To obtain the key, log in to the DWS console, click the username in the upper right corner, choose **My Credentials**, and click **Access Keys** in the navigation pane. On the displayed page, view the existing **Access Key ID (AK)**. To obtain both the AK and SK, click **Create Access Key** to create and download an access key.
- Step 3** Examine the formats of data to be exported and determine the values of data format parameters used for creating a foreign table. For details, see data format parameters.
- Step 4** Create an OBS table based on the parameter settings in the preceding steps.
- End

Example 1

For example, in the DWS database, create a write-only foreign table with the **format** parameter as **text** to export text files. Set parameters as follows:

- **location**
The OBS path of the source data file has been obtained in [step 2](#) in [Planning Data Export](#).
For example, set **location** as follows:

```
location 'obs://mybucket/output_data/',
```
- **Access keys (AK and SK)**
 - Set **access_key** to the AK you have obtained.
 - Set **secret_access_key** to the SK you have obtained.

 **NOTE**

access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.
- **Data format parameters**
 - Set **format** to **TEXT**.
 - Set **encoding** to **UTF-8**.
 - Configure **encrypt**. Its default value is **off**.
 - Set **delimiter** to **|**.

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

NOTICE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext1;  
CREATE FOREIGN TABLE product_info_output_ext1  
(  
  c_bigint bigint,  
  c_char char(30),  
  c_varchar varchar(30),  
  c_nvarchar2 nvarchar2(30) ,  
  c_data date,  
  c_time time ,  
  c_test varchar(30))  
server gsmpp_server  
options (  
  LOCATION 'obs://mybucket/output_data/',  
  ACCESS_KEY 'access_key_value_to_be_replaced',  
  SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'  
  format 'text',  
  delimiter '|',  
  encoding 'utf-8',  
  encrypt 'on'  
)  
WRITE ONLY;
```

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

```
CREATE FOREIGN TABLE
```

Example 2:

For example, in the DWS database, create a write-only foreign table with the **format** parameter as **CSV** to export CSV files. Set parameters as follows:

- **location**

The OBS path of the source data file has been obtained in [step 2](#) in [Preparing Data on OBS](#).

For example, set **location** as follows:

```
location 'obs://mybucket/output_data/',
```

- **Access keys (AK and SK)**

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

 NOTE

access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

- **Data format parameters**

- Set **format** to **CSV**.
- Set **encoding** to **UTF-8**.
- Configure **encrypt**. Its default value is **off**.
- Set **delimiter** to **,**.
- Set **header** (whether the exported data file contains the header row).
Specifies whether a file contains a header with the names of each column in the file.

When exporting data from OBS, this parameter cannot be set to **true**. Use the default value **false**, indicating that the first row of the exported data file is not the header.

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

NOTICE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext2;  
CREATE FOREIGN TABLE product_info_output_ext2  
(  
    product_price          integer      not null,  
    product_id             char(30)     not null,  
    product_time           date          ,  
    product_level          char(10)     ,  
    product_name           varchar(200) ,  
    product_type1          varchar(20)  ,  
    product_type2          char(10)     ,  
    product_monthly_sales_cnt integer    ,  
    product_comment_time   date          ,  
    product_comment_num    integer      ,  
    product_comment_content varchar(200)  
)  
SERVER gsmpp_server  
OPTIONS(  
    location 'obs://mybucket/output_data/',  
    FORMAT 'CSV' ,  
    DELIMITER ',',  
    encoding 'utf8',  
    header 'false',  
    ACCESS_KEY 'access_key_value_to_be_replaced',  
    SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'  
)  
WRITE ONLY ;
```

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

```
CREATE FOREIGN TABLE
```

2.6.1.2.3 Exporting Data

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

Examples

- **Example 1:** Export data from table **product_info_output** to a data file through the **product_info_output_ext** foreign table.

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

If information similar to the following is displayed, the data has been exported.

```
INSERT 0 10
```

- **Example 2:** Export part of the data to a data file by specifying the filter condition **WHERE product_price>500**.

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
```

 NOTE

- The directory to be used for data storage must be empty, or the export will fail.
- Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the RAWTOHEX() function to convert it to the hexadecimal format before export.

2.6.1.2.4 Examples

Exporting a Table

Create two foreign tables and use them to export tables from a database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input-data1** and **/input-data2** for storing data files, and create data directories **/input-data1/data** and **/input-data2/data**, respectively, in the two buckets.

Step 2 On the DWS database, create the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS data server to receive data exported from the database.

OBS and the database are in the same region. The example DWS table to be exported is **tpcds.customer_address**.

Export information is set as follows:

- The source data file directories are **/input-data1/data/** and **/input-data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input-data1/data/** and **obs://input-data2/data/**, respectively.

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

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- Configure **encrypt**. Its default value is **off**.
- **access_key** is set to the AK you have obtained. (mandatory)
- **secret_access_key** is set to the SK you have obtained. (mandatory)

 NOTE

access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

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

NOTICE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
  ca_address_sk      integer      ,
  ca_address_id      char(16)     ,
  ca_street_number   char(10)     ,
  ca_street_name     varchar(60)  ,
  ca_street_type     char(15)     ,
  ca_suite_number    char(10)     ,
  ca_city            varchar(60)  ,
  ca_county          varchar(30)  ,
  ca_state           char(2)      ,
  ca_zip             char(10)     ,
  ca_country         varchar(20)  ,
  ca_gmt_offset      decimal(5,2) ,
  ca_location_type   char(20)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
  ca_address_sk      integer      ,
  ca_address_id      char(16)     ,
  ca_street_number   char(10)     ,
  ca_street_name     varchar(60)  ,
  ca_street_type     char(15)     ,
  ca_suite_number    char(10)     ,
  ca_city            varchar(60)  ,
  ca_county          varchar(30)  ,
  ca_state           char(2)      ,
  ca_zip             char(10)     ,
  ca_country         varchar(20)  ,
  ca_gmt_offset      decimal(5,2) ,
  ca_location_type   char(20)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

Step 3 In DWS, export the data table **tpcds.customer_address** to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** concurrently.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.customer_address;
```

 NOTE

The design of OBS foreign tables does not allow exporting files to a non-empty path. However, in concurrent export scenarios, multiple files are exported to the same path, causing an error.

Assume that a user concurrently exports data from the same table to the same OBS foreign table, and that one SQL statement is executed to export data when another SQL statement is being executed and has not generated any file on the OBS server. In this case, certain data is overwritten although both SQL statements are successfully executed. Therefore, you are advised not to concurrently export data to the same OBS foreign table.

----End

Concurrently Exporting Tables

Use the two foreign tables to export tables from the database to two buckets in OBS.

Step 1 Log in to the OBS data server through the management console. On the OBS server, create the buckets **/input-data1** and **/input-data2** for storing data files, and create data directories **/input-data1/data** and **/input-data2/data**, respectively, in the two buckets.

Step 2 In DWS, create foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** for the OBS server to receive exported data.

OBS and the database are in the same region. Tables to be exported are **tpcds.customer_address** and **tpcds.customer_demographics**.

Export information is set as follows:

- The source data file directories are **/input-data1/data/** and **/input-data2/data/**, so **location** of **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2** is set to **obs://input-data1/data/** and **obs://input-data2/data/**, respectively.

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

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- Configure **encrypt**. Its default value is **off**.
- **access_key** is set to the AK you have obtained. (mandatory)
- **secret_access_key** is set to the SK you have obtained. (mandatory)

 NOTE

access_key and **secret_access_key** have been obtained during user creation. Replace the italic part with the actual keys.

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

NOTICE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
CREATE FOREIGN TABLE tpcds.customer_address_ext1
(
  ca_address_sk      integer      ,
  ca_address_id      char(16)     ,
  ca_street_number   char(10)     ,
  ca_street_name      varchar(60) ,
  ca_street_type     char(15)     ,
  ca_suite_number    char(10)     ,
  ca_city            varchar(60)  ,
  ca_county          varchar(30)  ,
  ca_state           char(2)      ,
  ca_zip             char(10)     ,
  ca_country         varchar(20)  ,
  ca_gmt_offset      decimal(5,2) ,
  ca_location_type   char(20)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input-data1/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
CREATE FOREIGN TABLE tpcds.customer_address_ext2
(
  ca_address_sk      integer      ,
  ca_address_id      char(16)     ,
  ca_address_name     varchar(20)  ,
  ca_address_code     integer     ,
  ca_street_number   char(10)     ,
  ca_street_name      varchar(60) ,
  ca_street_type     char(15)     ,
  ca_suite_number    char(10)     ,
  ca_city            varchar(60)  ,
  ca_county          varchar(30)  ,
  ca_state           char(2)      ,
  ca_zip             char(10)     ,
  ca_country         varchar(20)  ,
  ca_gmt_offset      decimal(5,2)
)
SERVER gsmpp_server
OPTIONS(LOCATION 'obs://input_data2/data/',
FORMAT 'CSV',
ENCODING 'utf8',
DELIMITER E'\x08',
QUOTE E'\x1b',
ENCRYPT 'off',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced'
)Write Only;
```

Step 3 In DWS, export the data tables **tpcds.customer_address** and **tpcds.warehouse** in parallel to the foreign tables **tpcds.customer_address_ext1** and **tpcds.customer_address_ext2**, respectively.

```
INSERT INTO tpcds.customer_address_ext1 SELECT * FROM tpcds.customer_address;
INSERT INTO tpcds.customer_address_ext2 SELECT * FROM tpcds.warehouse;
```

----End

2.6.1.3 Exporting ORC and Parquet Data to OBS

2.6.1.3.1 Planning Data Export

For details about exporting data to OBS, see [Planning Data Export](#).

For the data types that can be exported to OBS, see [Table 2-29](#).

Table 2-29 Mapping between ORC or Parquet write-only foreign tables and Hive data types

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
1-byte integer	TINYINT	TINYINT (not recommended)	SMALLINT
		SMALLINT (recommended)	SMALLINT
2-byte integer	SMALLINT	SMALLINT	SMALLINT
4-byte integer	INTEGER, BINARY_INTEGER	INTEGER	INT
8-byte integer	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4, REAL	FLOAT4, REAL	FLOAT
Double-precision floating point number	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE
Scientific data type	DECIMAL, NUMERIC	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	<i>precision</i> ≤ 38: DECIMAL; <i>precision</i> > 38: STRING
Date type	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
Time type	TIME [(p)] [WITHOUT TIME ZONE], TIME [(p)] [WITH TIME ZONE]	TEXT	STRING
	TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
	INTERVAL DAY (l) TO SECOND (p), INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
Boolean type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n), CHARACTER(n), NCHAR(n)	CHAR(n), CHARACTER(n), NCHAR(n)	$n \leq 255$: CHAR(n); $n > 255$: STRING
VARCHAR type	VARCHAR(n), CHARACTER VARYING(n), VARCHAR2(n)	VARCHAR(n)	$n \leq 65535$: VARCHAR(n); $n > 65535$: STRING
	NVARCHAR2(n)	TEXT	STRING
String (large text object)	TEXT, CLOB	TEXT, CLOB	STRING
Binary type	BYTEA	BYTEA	BINARY
Monetary type	MONEY	NUMERIC	BIGINT

For details about HDFS data export or MRS configuration, see the *MapReduce Service User Guide*.

 NOTE

This section uses the ORC format as an example to describe how to export data. The method for exporting Parquet data is similar. Parquet data can be exported from clusters of version 9.1.0 or later.

2.6.1.3.2 Creating a Foreign Server

Create an OBS foreign server and an HDFS foreign server.

Create an OBS external server to define the OBS server information for the foreign table. The creation method varies by cluster version. For details, see [Creating a Foreign Server](#).

- For 8.2.0 and later versions, delegate access to OBS bucket data by creating an OBS data source on the OBS console.
- For versions before 8.2.0, you will need to manually create an OBS server. For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

For how to create an HDFS foreign server, see [Manually Creating a Foreign Server](#).

2.6.1.3.3 Creating a Foreign Table

After operations in [Creating a Foreign Server](#) are complete, create an OBS/HDFS write-only foreign table in the DWS database to access data stored in OBS/HDFS. The foreign table is write-only and can be used only for data export.

The syntax for creating a foreign table is as follows. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
    column_constraint [...]} ] ]
  table_constraint [, ...] [, ...] )
SERVER dfs_server
OPTIONS ( { option_name ' value ' } [, ...] )
[ {WRITE ONLY } ]
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table named *product_info_ext_obs*, set parameters in the syntax as follows:

- **table_name**
Specifies the name of the foreign table to be created.
- **Table column definitions**
 - **column_name**: specifies the name of a column in the foreign table.
 - **type_name**: specifies the data type of the column.Multiple columns are separate by commas (,).
- **SERVER dfs_server**

Specifies the foreign server name of the foreign table. This server must exist. The foreign table connects to OBS/HDFS to read data through the foreign server.

Enter the name of the foreign server created by following steps in [Creating a Foreign Server](#).

- **OPTIONS parameters**

These are parameters associated with the foreign table. The key parameters are as follows:

- **format**: specifies the format of the exported data file. The ORC and Parquet formats are supported.
- **foldername**: (mandatory) specifies the data source file directory in the foreign table. OBS: specifies the OBS path of the data source file. You only need to enter */Bucket name/Folder directory level/*. HDFS: specifies the path in the HDFS file system. This parameter is mandatory for the write-only foreign table.
- **encoding**: specifies the encoding of the data source file in the foreign table. The default value is **utf8**.

- **filesize**

Specifies the file size of a write-only foreign table, in MB. If this parameter is not specified, the file size in the distributed file system configuration is used by default. This syntax is available only for the write-only foreign table.

Value range: an integer ranging from 1 to 1024

 NOTE

The **filesize** parameter is valid only for the ORC-formatted write-only HDFS foreign table.

- **compression**

(Optional) Specifies the compression mode of files. This syntax is available only for the write-only foreign table.

Value range: **zlib**, **snappy**, and **lz4**

Specifies the compression mode of ORC files. The default value is **zlib**. Specifies the compression mode of Parquet files. The default value is **snappy**.

- **version**

(Optional) Specifies the ORC version number. This syntax is available only for the write-only foreign table.

Value range: Only **0.12** is supported. The default value is **0.12**.

- **dataencoding**

(Optional) Specifies the data code of the data table to be exported when the database code is different from the data code of the data table. For example, the database code is Latin-1, but the data in the exported data table is in UTF-8 format. If this parameter is not specified, the database encoding format is used by default. This syntax is valid only for the write-only HDFS foreign table.

Value range: data code types supported by the database encoding

 NOTE

The **dataencoding** parameter is valid only for the ORC-formatted write-only HDFS foreign table.

- **Other parameters in the syntax**

Other parameters are optional and can be configured as required. In this example, they do not need to be configured.

Based on the preceding settings, the command for creating the foreign table is as follows:

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
```

```
-- Create an OBS foreign table that does not contain partition columns. The foreign server associated with the table is obs_server, the file format on OBS corresponding to the table is ORC, and the data storage path on OBS is /mybucket/data/.
```

```
CREATE FOREIGN TABLE product_info_ext_obs
(
  product_price      integer      ,
  product_id         char(30)     ,
  product_time       date         ,
  product_level      char(10)     ,
  product_name       varchar(200) ,
  product_type1      varchar(20)  ,
  product_type2      char(10)     ,
  product_monthly_sales_cnt integer ,
  product_comment_time date       ,
  product_comment_num integer     ,
  product_comment_content varchar(200)
) SERVER obs_server
OPTIONS (
  format 'orc',
  foldername '/mybucket/demo.db/product_info_orc',
  compression 'snappy',
  version '0.12'
) Write Only;
```

2.6.1.3.4 Exporting Data

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

Examples

- **Example 1:** Export data from table **product_info_output** to a data file using the **product_info_output_ext** foreign table.

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

If information similar to the following is displayed, the data has been exported.

```
INSERT 0 10
```

- **Example 2:** Export part of the data to a data file by specifying the filter condition **WHERE product_price>500**.

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;
```

 NOTE

Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. As a result, you need to use the **RAWTOHEX()** function to convert it to the hexadecimal format before export.

2.6.2 Exporting ORC and Parquet Data to MRS

2.6.2.1 Overview

DWS allows you to export ORC and Parquet data to MRS using an HDFS foreign table. You can specify the export mode and export data format in the foreign table. Data is exported from DWS in parallel using multiple DNs and stored in HDFS. In this way, the overall export performance is improved.

- The CN only plans data export tasks and delivers the tasks to DNs for execution. In this case, the CN is released to process external requests.
- Every DN is involved in data export, and the computing capabilities and bandwidths of all the DNs are fully leveraged to export data.
- Multiple HDFS servers can export data concurrently. The export path can be empty. The naming rule of the path must be the same as that of the exported file.
- MRS connects to cluster nodes. The export rate is affected by the network bandwidth.
- Data files in the ORC or Parquet format are supported.

NOTE

This section uses the ORC format as an example to describe how to export data. The method for exporting Parquet data is similar. Parquet data can be exported from clusters of version 9.1.0 or later.

Naming Rules of Exported Files

The rules for naming ORC and Parquet data files exported from DWS are as follows:

1. Data exported to MRS (HDFS): When data is exported from a DN, the data is stored in HDFS in the segment format. The file is named in the format of **mpp_Database name_Schema name_Table name_Node name_n_UUID.Data format**. *n* is a natural number starting from 0 in ascending order, for example, 0, 1, 2, 3. The UUID should be a standard one, comprising of 32 hexadecimal characters and divided into five segments by hyphens (-). The format should be 8-4-4-4-12.
2. You are advised to export data from different clusters or databases to different paths. The maximum size of a single file in ORC or Parquet format is about 256 MB. (This is a soft constraint and may exceed a little in actual services.)
3. After the export is complete, the **_SUCCESS** file is generated.

2.6.2.2 Planning Data Export

For details about the data types that can be exported to MRS, see [Table 2-30](#).

Table 2-30 Mapping between ORC or Parquet write-only foreign tables and Hive data types

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
1-byte integer	TINYINT	TINYINT (not recommended)	SMALLINT
		SMALLINT (recommended)	SMALLINT
2-byte integer	SMALLINT	SMALLINT	SMALLINT
4-byte integer	INTEGER, BINARY_INTEGER	INTEGER	INT
8-byte integer	BIGINT	BIGINT	BIGINT
Single-precision floating point number	FLOAT4, REAL	FLOAT4, REAL	FLOAT
Double-precision floating point number	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE PRECISION, FLOAT8, BINARY_DOUBLE	DOUBLE
Scientific data type	DECIMAL, NUMERIC	DECIMAL[p (,s)] (The maximum precision can reach up to 38.)	<i>precision</i> ≤ 38: DECIMAL; <i>precision</i> > 38: STRING
Date type	DATE	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
Time type	TIME [(p)] [WITHOUT TIME ZONE], TIME [(p)] [WITH TIME ZONE]	TEXT	STRING

Type	DWS Internal Table Type (Data Source Table)	DWS Write-Only Foreign Table Type	Hive Table Type
	TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], SMALLDATETIME	TIMESTAMP[(p)] [WITHOUT TIME ZONE]	TIMESTAMP
	INTERVAL DAY (l) TO SECOND (p), INTERVAL [FIELDS] [(p)]	VARCHAR(n)	VARCHAR(n)
Boolean type	BOOLEAN	BOOLEAN	BOOLEAN
CHAR type	CHAR(n), CHARACTER(n), NCHAR(n)	CHAR(n), CHARACTER(n), NCHAR(n)	$n \leq 255$: CHAR(n); $n > 255$: STRING
VARCHAR type	VARCHAR(n), CHARACTER VARYING(n), VARCHAR2(n)	VARCHAR(n)	$n \leq 65535$: VARCHAR(n); $n > 65535$: STRING
	NVARCHAR2(n)	TEXT	STRING
String (large text object)	TEXT, CLOB	TEXT, CLOB	STRING
Binary type	BYTEA	BYTEA	BINARY
Monetary type	MONEY	NUMERIC	BIGINT

For details about HDFS data export or MRS configuration, see the .

2.6.2.3 Creating a Foreign Server

To create a foreign server for HDFS, you need to create a user and database and provide the necessary permissions for the foreign table.

Creating a User and a Database and Granting the User Foreign Table Permissions

In the following example, a common user **dbuser** and a database **mydatabase** are created. Then, an administrator is used to grant foreign table permissions to user **dbuser**.

- Step 1** Connect to the default database **gaussdb** as a database administrator through the database client tool provided by DWS.

For example, use the **gsql** client to connect to the database by running the following command:

```
gsql -d gaussdb -h 192.168.2.30 -U dbadmin -p 8000 -W password -r
```

- Step 2** Create a common user and use it to create a database.

Create a user named **dbuser** that has the permission to create databases.

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

Switch to the created user.

```
SET ROLE dbuser PASSWORD 'password';
```

Run the following command to create a database:

```
CREATE DATABASE mydatabase;
```

Query the database.

```
SELECT * FROM pg_database;
```

The database is successfully created if the returned result contains information about **mydatabase**.

datname	datdba	encoding	datcollate	datctype	datistemplate	dataallowconn	datconnlimit	datacl
template1	10	0	C	C	t	t	-1	14146 1351
1663	ORA		{=c/Ruby,Ruby=CTc/Ruby}					
template0	10	0	C	C	t	f	-1	14146 1350
1663	ORA		{=c/Ruby,Ruby=CTc/Ruby}					
gaussdb	10	0	C	C	f	t	-1	14146 1352
1663	ORA		{=Tc/Ruby,Ruby=CTc/Ruby,chaojun=C/Ruby,huobinru=C/Ruby}					
mydatabase	17000	0	C	C	f	t	-1	14146 1351
1663	ORA							

(4 rows)

- Step 3** Grant the permissions for creating foreign servers and using foreign tables to a common user as the administrator.

Use the connection to create a database as a database administrator.

You can use the **gsql** client to run the following command, switching to an administrator user, and connect to the new database:

```
\c mydatabase dbadmin;
```

Enter the password as prompted.

 NOTE

Note that you must use the administrator account to connect to the database where a foreign server is to be created and foreign tables are used; and then grant permissions to the common user.

By default, only system administrators can create foreign servers. Common users can create foreign servers only after being authorized. Run the following command to grant the permission:

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
```

The name of **FOREIGN DATA WRAPPER** must be **hdfs_fdw**. **dbuser** is the username for creating **SERVER**.

Run the following command to grant the user the permission to use foreign tables:

```
ALTER USER dbuser USEFT;
```

Query for the user.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
       r.rolcreatorole, r.rolcreatedb, r.rolcanlogin,
       r.rolconlimit, r.rolvalidbegin, r.rolvaliduntil,
       ARRAY(SELECT b.rolname
             FROM pg_catalog.pg_auth_members m
             JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
             WHERE m.member = r.oid) as memberof
       , r.rolreplication
       , r.rolauditadmin
       , r.rolsystemadmin
       , r.roluseft
FROM pg_catalog.pg_roles r
ORDER BY 1;
```

The authorization is successful if the **dbuser** information in the returned result contains the **UseFT** permission.

rolname	rolsuper	rolinherit	rolcreatorole	rolcreatedb	rolcanlogin	rolconlimit	rolvalidbegin	rolvaliduntil	memberof	rolreplication	rolauditadmin	rolsystemadmin	roluseft
dbuser	f	t	f	t	t	-1			{}	f			f
lily	f	t	f	f	t	-1			{}	f			f
Ruby	t	t	t	t	t	-1			{}	t			t

----End

Manually Creating a Foreign Server

Step 1 Connect to the default database **postgres** as a database administrator through the database client tool provided by DWS.

You can use the **gsql** client to log in to the database in either of the following ways:

You can use either of the following methods to create the connection:

- If you have logged in to the **gsql** client, run the following command to switch the database and user:

```
\c postgres dbadmin;
```

Enter the password as prompted.

- If you have not logged in to the gsql client or have exited the gsql client by running the \q command, run the following command to reconnect to it:
gsql -d postgres -h 192.168.2.30 -U dbadmin -p 8000 -W password -r

Step 2 Run the following command to query the information about the foreign server that is automatically created:

```
SELECT * FROM pg_foreign_server;
```

The returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
-----+-----+-----+-----+-----+-----					
gsmpp_server	10	13673			
-----+-----+-----+-----+-----+-----					
{"address=192.168.1.245:25000,192.168.1.218:25000",hdfscfgpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}					
(2 rows)					

In the query result, each row contains the information about a foreign server. The foreign server associated with the MRS data source connection contains the following information:

- The value of **srvname** contains **hdfs_server** and the ID of the MRS cluster, which is the same as the MRS ID in the cluster list on the MRS management console.
- The **address** parameter in the **srvoptions** field contains the IP addresses and ports of the active and standby nodes in the MRS cluster.

You can find the foreign server you want based on the above information and record the values of its **srvname** and **srvoptions**.

Step 3 Switch to the user who is about to create a foreign server to connect to the corresponding database.

In this example, run the following command to use common user **dbuser** created in [Creating a User and a Database and Granting the User Foreign Table Permissions](#) to connect to **mydatabase** created by the user:

```
\c mydatabase dbuser;
```

Step 4 Create a foreign server.

For details about the syntax for creating an external server, see "CREATE SERVER" in *Data Warehouse Service (DWS) SQL Syntax Reference*. For example:

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca FOREIGN DATA  
WRAPPER HDFS_FDW  
OPTIONS  
(  
  address '192.168.1.245:25000,192.168.1.218:25000',  
  hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',  
  type 'hdfs'  
);
```

Mandatory parameters are described as follows:

- Name of the foreign server*

You can customize a name.

In this example, specify the name to the value of the **srvname** field recorded in [Step 2](#), such as **hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca**.

Resources in different databases are isolated. Therefore, the names of foreign servers in different databases can be the same.

- **FOREIGN DATA WRAPPER**

This parameter can only be set to **HDFS_FDW**, which already exists in the database.

- **OPTIONS** parameters

Set the following parameters to the values under **srvoptions** recorded in [Step 2](#).

- address

Specifies the IP address and port number of the primary and standby nodes of the HDFS cluster.

- hdfsconfigpath

Specifies the configuration file path of the HDFS cluster. This parameter is available only when **type** is **HDFS**. You can set only one path.

- type

Its value is **hdfs**, which indicates that **HDFS_FDW** connects to HDFS.

Step 5 View the foreign server.

```
SELECT * FROM pg_foreign_server WHERE
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

The server is successfully created if the returned result is as follows:

srvname	srvowner	srvfdw	srvtype	srversion	srvacl
hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca					

```
{ "address=192.168.1.245:25000,192.168.1.218:25000",hdfsconfigpath=/MRS/8f79ada0-d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

----End

2.6.2.4 Creating a Foreign Table

After operations in [Creating a Foreign Server](#) are complete, create an HDFS write-only foreign table in the DWS database to access data stored in HDFS. The foreign table is write-only and can be used only for data export.

The syntax for creating a foreign table is as follows. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
    column_constraint [...] } ] |
  table_constraint [, ...] [, ...] ] )
SERVER dfs_server
OPTIONS ( { option_name ' value ' } [, ...] )
[ {WRITE ONLY } ]
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}
[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

For example, when creating a foreign table *product_info_ext_obs*, configure the parameters in the syntax as follows.

- **table_name**
Specifies the name of the foreign table.
- **Table column definitions**
 - **column_name**: specifies the name of a column in the foreign table.
 - **type_name**: specifies the data type of the column.Multiple columns are separate by commas (,).
- **SERVER dfs_server**
Specifies the foreign server name of the foreign table. This server must exist. The foreign table connects to OBS/HDFS to read data through the foreign server.
Enter the name of the foreign server created in [Creating a Foreign Server](#).
- **OPTIONS parameters**
These parameters are associated with the foreign table. The key parameters are as follows:
 - **format**: specifies the format of the exported data file. The ORC and Parquet formats are supported.
 - **foldername**: specifies the directory of the data source file in the foreign table, that is, the corresponding file directory in HDFS. This parameter is mandatory for write-only foreign tables and optional for read-only foreign tables.
 - **encoding**: specifies the encoding format of the data source file in the foreign table. The default value is **utf8**.
 - **filesize**
(Optional) Specifies the file size of a write-only foreign table. If this parameter is not specified, the file size in the distributed file system is used by default. This syntax is available only for the write-only foreign table.
Value range: an integer ranging from 1 to 1024

 **NOTE**
The **filesize** parameter is valid only for the write-only HDFS foreign table in ORC format.

 - **compression**
(Optional) Specifies the compression mode of files. This syntax is available only for the write-only foreign table.
Value range: **zlib**, **snappy**, and **lz4**.
Specifies the compression mode of ORC files. The default value is **zlib**.
Specifies the compression mode of Parquet files. The default value is **snappy**.
 - **version**
(Optional) Specifies the ORC version number. This syntax is available only for the write-only foreign table.
Value range: Only **0.12** is supported. The default value is **0.12**.
 - **dataencoding**
(Optional) Specifies the data encoding of the data table to be exported when the database encoding is different from the data encoding of the

data table. For example, the database encoding is Latin-1, but the data encoding of the exported data table is in UTF-8 format. If this parameter is not specified, the database encoding is used by default. This syntax is valid only for the write-only HDFS foreign table.

Value range: data encoding types supported by the database encoding

NOTE

The **dataencoding** parameter is valid only for the write-only HDFS foreign table in ORC format.

- **Other parameters in the syntax**

Other parameters are optional and can be configured as required. In this example, they do not need to be configured. For details, see "CREATE FOREIGN TABLE (SQL on Hadoop or OBS)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

Based on the preceding settings, the command for creating the foreign table is as follows:

```
DROP FOREIGN TABLE IF EXISTS product_info_ext_obs;
```

-- Create an OBS foreign table that does not contain partition columns. The foreign server associated with the table is **hdfs_server**, the format of the file on HDFS corresponding to the table is ORC, and the data storage path on OBS is **/user/hive/warehouse/product_info_orc/**.

```
CREATE FOREIGN TABLE product_info_ext_obs
(
  product_price      integer      ,
  product_id         char(30)     ,
  product_time       date         ,
  product_level      char(10)     ,
  product_name       varchar(200) ,
  product_type1      varchar(20)  ,
  product_type2      char(10)     ,
  product_monthly_sales_cnt integer ,
  product_comment_time date       ,
  product_comment_num integer     ,
  product_comment_content varchar(200)
) SERVER obs_server
OPTIONS (
  format 'orc',
  foldername '/user/hive/warehouse/product_info_orc/',
  compression 'snappy',
  version '0.12'
) Write Only;
```

2.6.2.5 Exporting Data

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

Examples

- **Example 1:** Export data from table **product_info_output** to a data file using the **product_info_output_ext** foreign table.

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

If information similar to the following is displayed, the data has been exported.

```
INSERT 0 10
```

- **Example 2:** Export part of the data to a data file by specifying the filter condition **WHERE** *product_price*>500.
`INSERT INTO product_info_output_ext SELECT * FROM product_info_output WHERE product_price>500;`

 NOTE

Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. As a result, you need to use the **RAWTOHEX()** function to convert it to the hexadecimal format before export.

2.6.3 Using GDS to Export Data to a Remote Server

2.6.3.1 Exporting Data In Parallel Using GDS

In high-concurrency scenarios, you can use GDS to export data from a database to a common file system.

In the current GDS version, data can be exported from a database to a pipe file.

- When the local disk space of the GDS user is insufficient:
 - The data exported from GDS is compressed using the pipe to occupy less disk space.
 - The exported data is transferred through the pipe to the HDFS server for storage.
- If you need to cleanse data before exporting data:
 - You can compile programs as needed and read streaming data from pipes in real time.

 NOTE

- The current version does not support data export through GDS in SSL mode. Do not use GDS in SSL mode.
- All pipe files mentioned in this section refer to named pipes on Linux.
- To ensure the correctness of data import or export using GDS, you need to import or export data in the same compatibility mode.

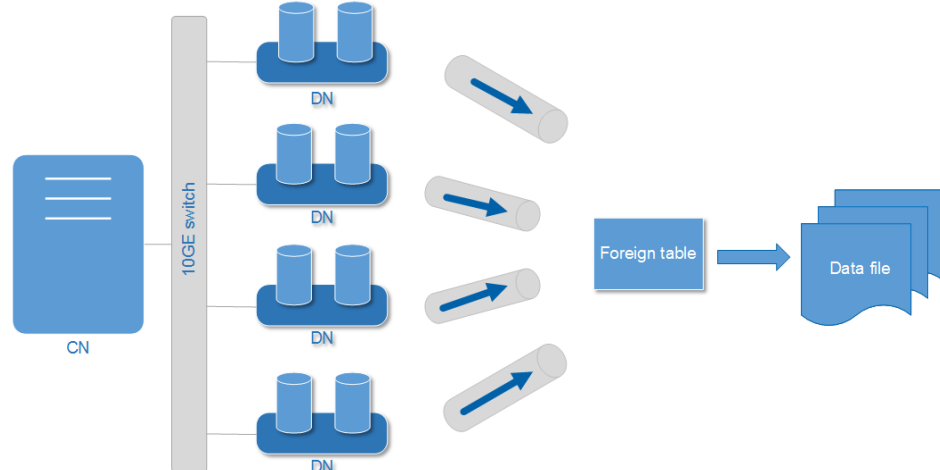
If data is imported or exported in MySQL compatibility mode, it can only be exported or imported in the same mode.

Overview

Using foreign tables: A GDS foreign table specifies the exported file format and export mode. Data is exported in parallel through multiple DNs from the database to data files, which improves the overall data export performance. The data files cannot be directly exported to HDFS.

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

Figure 2-9 Exporting data using foreign tables



Related Concepts

- **Data file:** A TEXT, CSV, or FIXED file that stores data exported from the DWS database.
- **Foreign table:** A table that stores information, such as the format, location, and encoding format of a data file.
- **GDS:** A data service tool. To export data, deploy it on the server where data files are stored.
- **Table:** Tables in the database, including row-store tables and column-store tables. Data in the data files is exported from these tables.
- **Remote mode:** Service data in a cluster is exported to hosts outside the cluster.

Exporting a Schema

Data can be exported to DWS in **Remote** mode.

- **Remote mode:** Service data in a cluster is exported to hosts outside the cluster.
 - In this mode, multiple GDSs are used to concurrently export data. One GDS can export data for only one cluster at a time.
 - The data export rate of a GDS that resides on the same intranet as cluster nodes is limited by the network bandwidth. A 10GE configuration is recommended.
 - Data files in TEXT, FIXED, or CSV format are supported. The size of data in a single row must be less than 1 GB.

Data Export Process

Figure 2-10 Concurrent data export

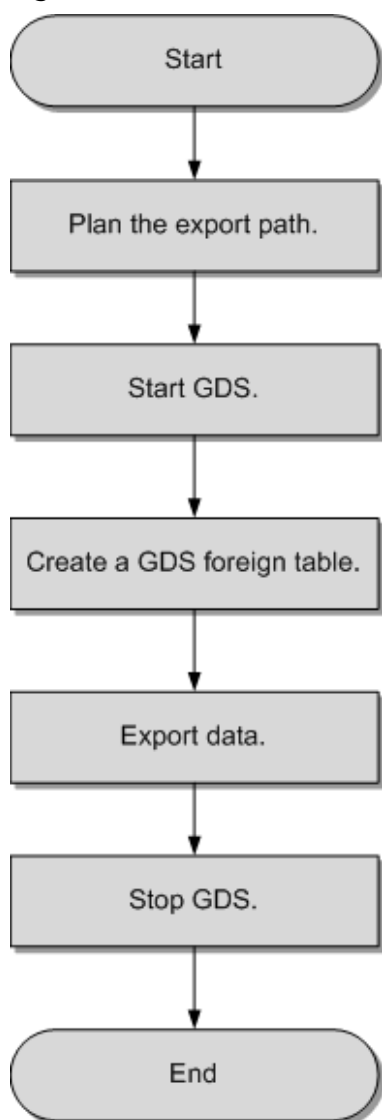


Table 2-31 Process description

Process	Description	Subtask
Plan data export.	Prepare data to be exported and plan the export path for the mode to be selected. For details, see Planning Data Export .	-

Process	Description	Subtask
Start GDS.	If the Remote mode is selected, install, configure, and start GDS on data servers. For details, see Installing, Configuring, and Starting GDS .	-
Create a foreign table,	Create a foreign table to help GDS specify information about a data file. The foreign table stores information, such as the location, format, encoding, and inter-data delimiter of a data file. For details, see Creating a GDS Foreign Table .	-
Export data.	After the foreign table is created, run the INSERT statement to efficiently export data to data files. For details, see Exporting Data .	-
Stop GDS.	Stop GDS after data is exported. For details, see Stopping GDS .	-

2.6.3.2 Planning Data Export

Scenarios

Before you use GDS to export data from a cluster, prepare data to be exported and plan the export path.

Planning an Export Path

- **Remote** mode

Step 1 Log in to the GDS data server as user **root** and create the **/output_data** directory for storing data files.

```
mkdir -p /output_data
```

Step 2 (Optional) Create a user and the user group to which it belongs. This user is used to start GDS and must have the write permission on the directory for storing data files.

```
groupadd gdsgrp  
useradd -g gdsgrp gdsuser
```

If the following information is displayed, the user and user group already exist. Skip this step.

```
useradd: Account 'gdsuser' already exists.  
groupadd: Group 'gdsgrp' already exists.
```

Step 3 Change the directory owner to **gdsuser**.

```
chown -R gdsuser:gdsgrp /output_data
```

----End

2.6.3.3 Installing, Configuring, and Starting GDS

GDS is a data service tool provided by DWS. Using the foreign table mechanism, this tool helps export data at a high speed.

To install, configure, and start GDS, perform the following steps. For details, see [Installing, Configuring, and Starting GDS](#).

Procedure

- Step 1** Log in as user **root** to the data server where GDS is to be installed and run the following command to create the directory for storing the GDS package:

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

- Step 2** Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_8.x.x_suse_x64.zip** to the directory created in the previous step.

- Step 3** (Optional) If SSL is used, upload the SSL certificates to the directory created in [Step 1](#).

- Step 4** Go to the directory and extract the package.

```
cd /opt/bin/dws
unzip dws_client_8.x.x_suse_x64.zip
```

- Step 5** Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

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

- Step 6** Change the owner of the GDS package directory and source data file directory to the GDS user.

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

- Step 7** Switch to user **gds_user**.

```
su - gds_user
```

If the current cluster version is 8.0.x or earlier, skip [Step 8](#) and go to [Step 9](#).

If the current cluster version is 8.1.x, go to the next step.

- Step 8** Execute the script on which the environment depends (applicable only to version 8.1.x).

```
cd /opt/bin/dws/gds/bin
source gds_env
```

- Step 9** Start GDS.

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

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

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

The first method is recommended when you do not need to import data again. The second method is recommended when you need to import data regularly.

- Method 1: Run the **gds** command to start GDS.

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

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

Example:

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

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

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

Example:

Run the following commands to upload the SSL certificates mentioned in [Step 3](#) to **/opt/bin**:

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

Replace the information in italic as required.

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

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

- **--enable-ssl**: enables SSL for data transmission.
- **--ssl-dir** *Cert_file*: SSL certificate directory. Set this parameter to the certificate directory in [Step 3](#).
- For details about GDS parameters, see "GDS - Parallel Data Loader > gds" in the *Data Warehouse Service (DWS) Tool Guide*.

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

- a. Run the following command to go to the **config** directory of the GDS package and modify the **gds.conf** configuration file. For details about the parameters in the **gds.conf** configuration file, see [Table 2-14](#).

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

Example:

Configure the **gds.conf** file as follows:

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

Information in the configuration file is as follows:

- The data server IP address is **192.168.0.90**, and the GDS listening port is **5000**.
 - Data files are stored in **/input_data/**.
 - Error log files are stored in **/err**. This directory must be created by a user with GDS read and write permissions.
 - The size of a single data file is 100 MB.
 - The size of each error log file is 100 MB.
 - Logs are stored in the **/log/gds_log.txt** file. This directory must be created by a user with GDS read and write permissions.
 - Only nodes with the IP address **10.10.0.*** can be connected.
 - The GDS process runs in the background.
 - Data file directories are recursive.
 - The number of concurrent import threads is 2.
- b. Start GDS and check whether it has been started.

```
python3 gds_ctl.py start
```

Example:

```
cd /opt/bin/dws/gds/bin
python3 gds_ctl.py start
Start GDS gds1 [OK]
gds [options]:
-d dir      Set data directory.
-p port      Set GDS listening port.
  ip:port    Set GDS listening ip address and port.
-l log_file  Set log file.
-H secure_ip_range
              Set secure IP checklist in CIDR notation. Required for GDS to start.
-e dir      Set error log directory.
-E size      Set size of per error log segment.(0 < size < 1TB)
-S size      Set size of data segment.(1MB < size < 100TB)
-t worker_num Set number of worker thread in multi-thread mode, the upper limit is 200. If
without setting, the default value is 8.
-s status_file Enable GDS status report.
-D          Run the GDS as a daemon process.
-r          Read the working directory recursively.
-h          Display usage.
```

----End

2.6.3.4 Creating a GDS Foreign Table

Procedure

Step 1 Set the **location** parameter for the foreign table based on the path planned in [Planning Data Export](#).

- **Remote** mode

Set the **location** parameter to the URL of the directory that stores the data files.

- You do not need to specify a file name in the URL.
- When the number of data sources exported is fewer than the available paths, files are created for the surplus paths without writing any data.

For example:

The IP address of the GDS data server is 192.168.0.90. The listening port number set during GDS startup is 5000. The directory for storing data files is **/output_data**.

In this case, set the **location** parameter to **gsfs://192.168.0.90:5000/**.

 NOTE

- By setting **location** to a subdirectory, for example, **gsfs://192.168.0.90:5000/2019/11/**, you can export the same table to different directories based on the date.
- In the current version, the system checks if the **/output_data/2019/11** directory exists when an export task is executed. If it does not exist, the system creates it. During the export, files are written to this directory. In this way, you do not need to manually run the **mkdir -p /output_data/2019/11** command after creating or modifying a foreign table.

Step 2 Set data format parameters in the foreign table based on the planned data file formats.

Step 3 Create a GDS foreign table based on the parameter settings in the preceding steps.

----End

Example

- **Example:** Create the GDS foreign table **foreign_tpcds_reasons** for the source data. Data is to be exported as CSV files.

Data export mode settings are as follows:

The data server resides on the same intranet as the cluster. The IP address of the data server is 192.168.0.90. Data is to be exported as CSV files. The **Remote** mode is selected for parallel data export.

Assume that the directory for storing data files is **/output_data/** and the GDS listening port is 5000 when GDS is started. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

Data format parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.

- **delimiter** is set to **E'\x08'**.
- **quote** is set to **E'\x1b'**.
- **null** is set to an empty string without quotation marks.
- **escape** is set to the same value as that of **quote** by default.
- **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.
- **EOL** is set to **0X0A**.

The foreign table is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server
OPTIONS (LOCATION 'gsfs://192.168.0.90:5000/',
        FORMAT 'CSV',
        DELIMITER E'\x08',
        QUOTE E'\x1b',
        NULL '',
        EOL '0x0a'
)
WRITE ONLY;
```

2.6.3.5 Exporting Data

Prerequisites

Ensure that the IP addresses and ports of servers where CNs and DN are deployed can connect to those of the GDS server.

Syntax

Run the following command to export data:

```
INSERT INTO [Foreign table name] SELECT * FROM [Source table name];
```

NOTE

Create batch processing scripts to export data in parallel. The degree of parallelism depends on the server resource usage. You can test several tables and monitor resource usage to determine whether to increase or reduce the amount. Common resource monitoring commands include **top** for memory and CPU usage, **iostat** for I/O usage, and **sar** for networks. For details about application cases, see [Exporting Data Using Multiple Threads](#).

Examples

- **Example 1:** Export data from the **reason** table to data files through the **foreign_tpcds_reasons** foreign table.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;
```
- **Example 2:** Export part of the data to data files by specifying the filter condition **r_reason_sk =1**.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason WHERE r_reason_sk=1;
```
- **Example 3:** Data of a special type, such as RAW, is exported as a binary file, which cannot be recognized by the import tool. You need to use the **RAWTOHEX()** function to convert it to hexadecimal the format before export.

```
INSERT INTO foreign_tpcds_reasons SELECT RAWTOHEX(c) FROM tpcds.reason;
```


2.6.3.6 Stopping GDS

GDS is a data service tool provided by DWS that enables high-speed data export through coordination with external mechanisms.

If GDS is no longer used, perform the following steps to stop it:

Procedure

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

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

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

- a. Query the GDS process ID:

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

For example, the GDS process ID is 128954.

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

```
gds_user 128954 1 0 15:03 ? 00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -l /log/  
gds_log.txt -D  
gds_user 129003 118723 0 15:04 pts/0 00:00:00 grep gds
```

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

```
kill -9 128954
```

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

```
cd /opt/bin/dws/gds/bin  
python3 gds_ctl.py stop
```

----End

2.6.3.7 Examples of Exporting Data Using GDS

Exporting Data in Remote Mode

The data server and the cluster reside on the same intranet, the IP address of the data server is **192.168.0.90**, and data source files are in CSV format. In this scenario, data is exported in parallel in **Remote** mode.

To export data in parallel in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create user **gds_user** and its user group.

```
mkdir -p /output_data
```

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

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

3. Change the owner of the **/output_data** directory on the data server to **gds_user**.

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

4. Log in to the data server as user **gds_user** and start GDS.

The GDS installation path is **/opt/bin/dws/gds**. Exported data files are stored in **/output_data/**. The IP address of the data server is **192.168.0.90**. The GDS listening port is **5000**. GDS runs in daemon mode.

```
/opt/bin/dws/gds/bin/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```

5. In the database, create the foreign table **foreign_tpcds_reasons** for receiving data from the data server.

Data export mode settings are as follows:

- The directory for storing exported files is **/output_data/** and the GDS listening port is **5000** when GDS is started. The directory created for storing exported files is **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.

Data format parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **E'\x1b'**.
- **null** is set to an empty string without quotation marks.
- **escape** defaults to the value of **quote**.
- **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

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

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

6. In the database, export data to data files through the foreign table **foreign_tpcds_reasons**.

```
INSERT INTO foreign_tpcds_reasons SELECT * FROM tpcds.reason;
```
7. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is **128954**.

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

Exporting Data Using Multiple Threads

The data server and the cluster reside on the same intranet, the IP address of the data server is **192.168.0.90**, and data source files are in CSV format. In this scenario, data is concurrently exported to two target tables using multiple threads in **Remote** mode.

To concurrently export data using multiple threads in **Remote** mode, perform the following operations:

1. Log in to the GDS data server as user **root**, create the **/output_data** directory for storing data files, and create the database user and its user group.

```
mkdir -p /output_data  
groupadd gdsgrp  
useradd -g gdsgrp gds_user
```
2. Change the owner of the **/output_data** directory on the data server to **gds_user**.

```
chown -R gds_user:gdsgrp /output_data
```
3. Log in to the data server as user **gds_user** and start GDS.
The GDS installation path is **/opt/bin/dws/gds**. Exported data files are stored in **/output_data/**. The IP address of the data server is **192.168.0.90**. The GDS listening port is **5000**. GDS runs in daemon mode. The degree of parallelism is 2.

```
/opt/bin/dws/gds/bin/gds -d /output_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D -t 2
```
4. In DWS, create the foreign tables **foreign_tpcds_reasons1** and **foreign_tpcds_reasons2** for receiving data from the data server.
 - Data export mode settings are as follows:
 - The directory for storing exported files is **/output_data/** and the GDS listening port is **5000** when GDS is started. The directory created for storing exported files is **/output_data/**. Therefore, the **location** parameter is set to **gsfs://192.168.0.90:5000/**.
 - Data format parameter settings are as follows:
 - **format** is set to **CSV**.
 - **encoding** is set to **UTF-8**.
 - **delimiter** is set to **E'\x08'**.
 - **quote** is set to **E'\x1b'**.
 - **null** is set to an empty string without quotation marks.
 - **escape** defaults to the value of **quote**.
 - **header** is set to **false**, indicating that the first row is identified as a data row in an exported file.

Based on the preceding settings, the foreign table **foreign_tpcds_reasons1** is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons1  
(  
  r_reason_sk integer not null,  
  r_reason_id char(16) not null,  
  r_reason_desc char(100)  
)  
SERVER gsmpp_server  
OPTIONS  
(  
  LOCATION 'gsfs://192.168.0.90:5000/',  
  FORMAT 'CSV',  
  ENCODING 'utf8',  
  DELIMITER E'\x08',  
  QUOTE E'\x1b',
```

```
NULL "  
)  
WRITE ONLY;
```

Based on the preceding settings, the foreign table **foreign_tpcds_reasons2** is created using the following statement:

```
CREATE FOREIGN TABLE foreign_tpcds_reasons2  
(  
  r_reason_sk integer not null,  
  r_reason_id char(16) not null,  
  r_reason_desc char(100)  
)  
SERVER gsmpp_server  
OPTIONS  
(  
  LOCATION 'gsfs://192.168.0.90:5000/',  
  FORMAT 'CSV',  
  DELIMITER E'\x08',  
  QUOTE E'\x1b',  
  NULL "  
)  
WRITE ONLY;
```

5. Export data from table **reasons1** to foreign table **foreign_tpcds_reasons1** and from table **reasons2** to foreign table **foreign_tpcds_reasons2** in the database, and save the exported data to the **/output_data** directory.
INSERT INTO foreign_tpcds_reasons1 SELECT * FROM tpcds.reason;
INSERT INTO foreign_tpcds_reasons2 SELECT * FROM tpcds.reason;
6. After data export is complete, log in to the data server as user **gds_user** and stop GDS.

The GDS process ID is **128954**.

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

Exporting Data Through a Pipe

Step 1 Start GDS.

```
gds -d /***gds_data/ -D -p 192.168.0.1:7789 -l /***gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data.

1. Log in to the database, create an internal table, and write data to the table.

```
CREATE TABLE test_pipe( id integer not null, gender text not null, name text ) ;
```

```
INSERT INTO test_pipe values(1,2,'1111111111111111');  
INSERT INTO test_pipe values(2,2,'1111111111111111');  
INSERT INTO test_pipe values(3,2,'1111111111111111');  
INSERT INTO test_pipe values(4,2,'1111111111111111');  
INSERT 0 1
```

2. Create a write-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe_tw( id integer not null, age text not null, name text )  
SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/', FORMAT 'text', DELIMITER ',',  
NULL "", EOL '0x0a', file_type 'pipe', auto_create_pipe 'false') WRITE ONLY;
```

3. Execute the export statement. In this case, the statements are blocked.

```
INSERT INTO foreign_test_pipe_tw select * from test_pipe;
```

Step 3 Export data through the GDS pipes.

1. Log in to GDS and go to the GDS data directory.
`cd /***/gds_data/`
2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.
`mkfifo postgres_public_foreign_test_pipe_tw.pipe`

 **NOTE**

A pipe will be automatically cleared after an operation is complete. To perform another operation, create a pipe file again.
3. Read data from the pipe and write it to a new file.
`cat postgres_public_foreign_test_pipe_tw.pipe > postgres_public_foreign_test_pipe_tw.txt`
4. To compress the exported files, run the following command:
`gzip -9 -c < postgres_public_foreign_test_pipe_tw.pipe > out.gz`
5. To export the content from the pipe to the HDFS server, run the following command:
`cat postgres_public_foreign_test_pipe_tw.pipe | hdfs dfs -put - /user/hive/***/test_pipe.txt`

Step 4 Verify the exported data.

1. Check whether the exported file is correct.
`cat postgres_public_foreign_test_pipe_tw.txt`
3,2,1111111111111111
1,2,1111111111111111
2,2,1111111111111111
4,2,1111111111111111
2. View the compressed file.
`vim out.gz`
3,2,1111111111111111
1,2,1111111111111111
2,2,1111111111111111
4,2,1111111111111111
3. View the data exported to the HDFS server.
`hdfs dfs -cat /user/hive/***/test_pipe.txt`
3,2,1111111111111111
1,2,1111111111111111
2,2,1111111111111111
4,2,1111111111111111

----End

Exporting Data Through Multi-Process Pipes

GDS also supports importing and exporting data through multi-process pipes. That is, one foreign table corresponds to multiple GDSs.

The following takes exporting a local file as an example.

Step 1 Start multiple GDSs.

```
gds -d /***/gds_data/ -D -p 192.168.0.1:7789 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
gds -d /***/gds_data_1/ -D -p 192.168.0.1:7790 -l /***/gds_log/aa.log -H 0/0 -t 10 -D
```

If you need to set the timeout interval of a pipe, use the **--pipe-timeout** parameter.

Step 2 Export data.

1. Log in to the database and create an internal table.
`CREATE TABLE test_pipe (id integer not null, gender text not null, name text);`
2. Write data.
`INSERT INTO test_pipe values(1,2,'1111111111111111');`
`INSERT INTO test_pipe values(2,2,'1111111111111111');`

```
INSERT INTO test_pipe values(3,2,'1111111111111111');  
INSERT INTO test_pipe values(4,2,'1111111111111111');
```

3. Create a write-only foreign table.

```
CREATE FOREIGN TABLE foreign_test_pipe_tw( id integer not null, age text not null, name text )  
SERVER gsmpp_server OPTIONS (LOCATION 'gsfs://192.168.0.1:7789/|gsfs://192.168.0.1:7790/',  
FORMAT 'text', DELIMITER ',', NULL '', EOL '0x0a',file_type 'pipe', auto_create_pipe 'false') WRITE  
ONLY;
```

4. Execute the export statement. In this case, the statements are blocked.

```
INSERT INTO foreign_test_pipe_tw select * from test_pipe;
```

Step 3 Export data through the GDS pipes.

1. Log in to GDS and go to each GDS data directory.

```
cd /***/gds_data/  
cd /***/gds_data_1/
```

2. Create a pipe. If **auto_create_pipe** is set to **true**, skip this step.

```
mkfifo postgres_public_foreign_test_pipe_tw.pipe
```

3. Read each pipe and write the new file to the pipes.

```
cat postgres_public_foreign_test_pipe_tw.pipe > postgres_public_foreign_test_pipe_tw.txt
```

Step 4 Verify the exported data.

```
cat /***/gds_data/postgres_public_foreign_test_pipe_tw.txt  
3,2,1111111111111111  
cat /***/gds_data_1/postgres_public_foreign_test_pipe_tw.txt  
1,2,1111111111111111  
2,2,1111111111111111  
4,2,1111111111111111
```

----End

2.7 Other Operations

2.7.1 GDS Pipe FAQs

Precautions

- GDS supports concurrent import and export. The **gds -t** parameter is used to set the size of the thread pool and control the maximum number of concurrent working threads. But it does not accelerate a single SQL task. The default value of **gds -t** is **8**, and the upper limit is **200**. When using the pipe function to import and export data, ensure that the value of **-t** is greater than or equal to the number of concurrent services. In the dual-cluster interconnection scenario, the value of **-t** must be greater than or equal to twice the number of concurrent services.
- Data in pipes is deleted once read. Therefore, ensure that no other program except GDS reads data in the pipe during import or export. Otherwise, data may be lost, task errors may occur, or the exported files may be disordered.
- Concurrent import and export of foreign tables with the same location are not supported. That is, multiple threads of GDS cannot read or write pipe files at the same time.
- A single import or export task of GDS identifies only one pipe. Therefore, do not carry wildcard characters ({ } [] ?) in the location address set for the GDS foreign table. Example:

```
CREATE FOREIGN TABLE foreign_test_pipe_tr( like test_pipe ) SERVER gsmpp_server OPTIONS  
(LOCATION 'gsfs://192.168.0.1:7789/foreign_test_*', FORMAT 'text', DELIMITER ',', NULL "", EOL  
'0x0a',file_type 'pipe',auto_create_pipe 'false');
```

- When the **-r** recursion parameter is enabled for GDS, only one pipe can be identified. That is, GDS identifies only one pipe in the current data directory and does not recursively search for it. Therefore, the **-r** parameter does not take effect in the pipe import and export scenarios.
- CN retry is not supported during the import and export through a pipe, because GDS cannot control the operations performed by peer users and programs on pipes.
- During the import, if the peer program does not write data into the pipe for more than one hour, the import task times out and an error is reported.
- During the export, if the peer program does not read data from the pipe for more than one hour by default, the export task times out and an error is reported.
- Ensure that the GDS version and kernel version support the function of importing and exporting data through pipes.
- If the **auto_create_pipe** parameter of the foreign table is set to **true**, a delay may occur when GDS automatically creates a pipe. Before any operation on a pipe, check whether the automatically created pipe exists and whether it is a pipe file.
- Once an import or export task through a GDS pipe is complete, the pipe is automatically deleted. However, the pipe deletion is delayed, if you manually terminate an import or export task. In this situation, the pipe is deleted after the timeout interval expires.

Common Troubleshooting Methods:

- Issue 1: **"/***/postgres_public_foreign_test_pipe_tr.pipe" must be named pipe.**
Locating method: The type of the GDS foreign table **file_type** is pipe, but the operated file is a common file. Check whether the **postgres_public_foreign_test_pipe_tr.pipe** file is a pipe file.
- Issue 2: **could not open pipe "/***/postgres_public_foreign_test_pipe_tw.pipe" cause by Permission denied.**
Locating method: GDS does not have the permission to open the pipe file.
- Issue 3: **could not open source file /*****/postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for WRITING.**
Locating method: Opening the pipe times out when GDS is used to export data. This is because the pipe is not created within 300 seconds after **auto_create_pipe** is set to **false**, or the pipe is created but is not read by any program within 300 seconds.
- Issue 4: **could not open source file /*****/postgres_public_foreign_test_pipe_tw.pipe because timeout 300s for READING.**
Locating method: Opening the pipe times out when GDS is used to export data. This is because the pipe is not created within 300 seconds after **auto_create_pipe** is set to **false**, or the pipe is created but is not written by any program within 300 seconds.

- Issue 5: **could not poll writing source pipe file "/****/postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.**
Locating method: If the GDS does not receive any write event on the pipe within 300 seconds during data export, the pipe is not read for more than 300 seconds.
- Issue 6: **could not poll reading source pipe file "/****/postgres_public_foreign_test_pipe_tw.pipe" timeout 300s.**
Locating method: If the GDS does not receive any read event on the pipe within 300 seconds during data import, the pipe is not written for more than 300 seconds.
- Issue 7: **could not open pipe file "/***/postgres_public_foreign_test_pipe_tw.pipe" for "WRITING" with error No such device or address.**
Locating method: It indicates that the /***/postgres_public_foreign_test_pipe_tw.pipe file is not read by any program. As a result, GDS cannot open the pipe file by writing.

2.7.2 Checking for Data Skew

Scenarios

Data skew causes the query performance to deteriorate. Before importing all the data from a table consisting of over 10 million records, you are advised to import some of the data and check whether data skew occurs and whether the distribution keys need to be changed. Troubleshoot the problems if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported.

Background

DWS uses a massively parallel processing (MPP) system of the shared-nothing architecture. The MPP performs horizontal partitioning to store tuples in service data tables on all DN nodes using proper distribution policies.

The following user table distribution policies are supported:

- Replication: stores a full table on each DN. You are advised to use the replication mode for tables containing a small volume of data.
- Hash: A distribution key must be specified for a user table. If a record is inserted, the system performs hash computing based on values in the distribute column and then stores data on the related DN. You are advised to use the hash distribution policy for tables with a large volume of data.
- Round-robin: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. If no proper distribution column can be found in a table with a large amount of data in hash mode, you are advised to use the round-robin distribution policy.

If an inappropriate distribution key is used, data skew may occur when you use the hash policy. Check for data skew when you use the hash distribution policy so that data can be evenly distributed to each DN. You are advised to use the column with few replicated values as the distribution key.

Procedure

Step 1 Analyze data source features and select candidate distribution columns that have more distinct values and evenly distributed data.

Step 2 Select a candidate column from [Step 1](#) to create a target table.

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ] table_name  
({ column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]  
| table_constraint | LIKE source_table [ like_option [...] ] }  
[, ... ]) [ WITH ( {storage_parameter = value} [, ... ] ) ]  
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]  
[ COMPRESS | NOCOMPRESS ] [ TABLESPACE tablespace_name ]  
[ DISTRIBUTE BY { REPLICATION  
| ROUNDROBIN  
| { HASH ( column_name [,...] ) } } ];
```

Step 3 Import a small batch of data to the target table.

When importing a single data file, you can evenly split this file and import a part of it to check for the data skew in the target table.

Step 4 Check for data skew. (Replace *table_name* with the actual table name.)

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP  
BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

Step 5 If the data distribution deviation is less than 10% across DNs, data is evenly distributed and an appropriate distribution key has been selected. Delete the small batch of imported data and import full data to complete data migration.

If data distribution deviation across DNs is greater than or equal to 10%, data skew occurs. Remove this distribution key from the candidates in [Step 1](#), delete the target table, and repeat [Step 2](#) through [Step 5](#).

NOTE

The data distribution deviation indicates the difference between the actual data volume on DNs and the average data volume on DNs. You can view the distribution difference in the [PGXC_GET_TABLE_SKEWNESS](#) view.

Step 6 (Optional) If you fail to select an appropriate distribution key after performing the preceding steps, select multiple columns from the candidates as distribution keys.

----End

Examples

Assume you want to select an appropriate distribution key for the **staffs** table.

1. Analyze the source data for the **staffs** table and select the **staff_ID**, **FIRST_NAME**, and **LAST_NAME** columns as candidate distribution keys.
2. Select the **staff_ID** column as the distribution key and create the target table **staffs**.

```
CREATE TABLE staffs  
(  
  staff_ID    NUMBER(6) not null,  
  FIRST_NAME  VARCHAR2(20),  
  LAST_NAME   VARCHAR2(25),  
  EMAIL       VARCHAR2(25),  
  PHONE_NUMBER VARCHAR2(20),  
  HIRE_DATE   DATE,  
  employment_ID VARCHAR2(10),
```

```
SALARY      NUMBER(8,2),
COMMISSION_PCT NUMBER(2,2),
MANAGER_ID  NUMBER(6),
section_ID  NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID);
```

3. Import a small batch of data to the target table **staffs**.

There are eight DNs in the cluster based on the following query, and you are advised to import 80,000 records.

```
SELECT count(*) FROM pgxc_node where node_type='D';
count
-----
      8
(1 row)
```

4. Verify the data skew of the target table **staffs** whose distribution key is **staff_ID**:

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY
xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
count | node_name
-----+-----
11010 | datanode4
10000 | datanode3
12001 | datanode2
8995  | datanode1
10000 | datanode5
7999  | datanode6
9995  | datanode7
10000 | datanode8
(8 rows)
```

5. The preceding query result indicates that the distribution deviation across DNs is greater than 10%. The data skew occurs. Therefore, delete **staff_ID** from the distribution key candidates and delete the **staffs** table.

```
DROP TABLE staffs;
```

6. Use **staff_ID**, **FIRST_NAME**, and **LAST_NAME** as distribution keys and create the target table **staffs**.

```
CREATE TABLE staffs
(
  staff_ID      NUMBER(6) not null,
  FIRST_NAME    VARCHAR2(20),
  LAST_NAME     VARCHAR2(25),
  EMAIL         VARCHAR2(25),
  PHONE_NUMBER  VARCHAR2(20),
  HIRE_DATE     DATE,
  employment_ID VARCHAR2(10),
  SALARY        NUMBER(8,2),
  COMMISSION_PCT NUMBER(2,2),
  MANAGER_ID    NUMBER(6),
  section_ID    NUMBER(4)
)
DISTRIBUTE BY hash(staff_ID,FIRST_NAME,LAST_NAME);
```

7. Verify the data skew of the target table **staffs** whose distribution keys are **staff_ID**, **FIRST_NAME**, and **LAST_NAME**.

```
SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY
xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
count | node_name
-----+-----
10010 | datanode4
10000 | datanode3
10001 | datanode2
9995  | datanode1
10000 | datanode5
9999  | datanode6
9995  | datanode7
```

```
10000 | datanode8  
(8 rows)
```

8. The preceding query result indicates that the data deviation across DNs is less than 10%. The data is evenly distributed and the appropriate distribution keys have been selected.
9. Delete the imported small-batch data.

```
TRUNCATE TABLE staffs;
```
10. Import the full data to complete data migration.

2.7.3 Analyzing a Table

The execution plan generator needs to use table statistics to generate the most effective query execution plan to improve query performance. After data is imported, you are advised to run the **ANALYZE** statement to update table statistics. The statistics are stored in the **PG_STATISTIC** system catalog.

Analyzing a Table

ANALYZE supports row-store, column-store, HDFS, and OBS tables in ORC or CARBONDATA format. **ANALYZE** can also collect statistics about specified columns of a local table.

Do **ANALYZE** to the **product_info** table.

```
ANALYZE product_info;
```

Automatically Analyzing a Table

DWS provides automatic table analysis for the following two scenarios.

- If **ANALYZE** is triggered because a query contains a table that has no statistics or a table whose amount of data modification reaches the threshold, and the execution plan does not use Fast Query Shipping (FQS), the GUC parameter **autoanalyze** is used to control the automatic collection of table statistics. In this case, a better execution plan is generated based on the collected statistics.
- When autovacuum is set to **on**, the system periodically starts the **autovacuum** thread to automatically collect statistics on tables whose amount of data modification reaches the threshold for triggering **ANALYZE** in the background.

Table 2-32 Automatically Analyzing a Table

Trigger Mode	Trigger Condition	Scaling Frequency	Control Parameter	Remarks
Synchronization	Statistics are completely missing.	At each query execution	autoanalyze	Statistics are cleared when the primary table is truncated.

Trigger Mode	Trigger Condition	Scaling Frequency	Control Parameter	Remarks
Synchronization	The amount of modified data reaches the ANALYZE threshold.	At each query execution	autoanalyze	ANALYZE is triggered before the optimal plan is determined.
Asynchronization	The amount of modified data reaches the ANALYZE threshold.	Autovacuum thread polling check	autovacuum_mode, autovacuum_naptime	The lock times out in 2 seconds, and the execution times out in 5 minutes.

NOTICE

- The autoanalyze function supports only the default sampling mode and not the percentage sampling mode.
 - The autoanalyze function does not collect multi-column statistics, which only supports percentage sampling.
 - **AUTOANALYZE** is triggered because a query contains a table that has no statistics or a table whose amount of data modification reaches the threshold. In this case, **AUTOANALYZE** cannot be triggered for foreign tables or temporary tables with the **ON COMMIT [DELETE ROWS | DROP]** option.
 - If the amount of data modification reaches the threshold for triggering **ANALYZE**, the amount of data modification exceeds $\text{autovacuum_analyze_threshold} + \text{autovacuum_analyze_scale_factor} * \text{reltuples}$. *reltuples* indicates the estimated number of rows in the table recorded in *pg_class*.
 - The autoanalyze function triggered by a scheduled **autovacuum** thread supports only row-store and column-store tables. It does not support foreign tables, HDFS tables, OBS foreign tables, temporary tables, unlogged tables, or toast tables.
 - When **ANALYZE** is triggered during a query, a level-4 lock is added to all partitions in the partitioned table. The lock is released only after the transaction containing the query is committed. The level-4 lock does not block adding, deletion, modification, and query operations, but blocks partition modification operations such as **TRUNCATE**. You can set **object_mtime_record_mode** to **disable_partition** to release the partition locks in advance.
 - The autovacuum function also depends on the following two GUC parameters in addition to **autovacuum**:
 - **track_counts** must be set to **on** to enable statistics collection about the database.
 - **autovacuum_max_workers** must be set to a value greater than 0 to specify the maximum number of concurrent autovacuum threads.
-

3 DWS Development Design Proposal

3.1 Overview

Objective

This document outlines the rules for design and development that need to be followed when developing the DWS database. The objective is to enhance development efficiency and ensure the continuity and stability of the service.

Application Scope

These specifications apply to all DWS self-development scenarios, including designing and developing applications and database services.

Terms

Rule: a mandatory requirement that must be followed during database design and development.

Suggestion: an option that you need to consider for the design and development process.

Description: a detailed explanation of a rule or suggestion.

Overall Development and Design Specifications

The table below provides a list of development and design specifications that must be followed during DWS development. You can click the links to access the corresponding rules for more details.

Table 3-1 DWS development and design specifications

N o.	Category		Rule/Suggestion
1	Conn ectio n man agem ent regul ation s	-	Rule 1.1: Configuring Load Balancing for DWS Clusters
2			Rule 1.2: Ending the Database Connection After Necessary Operations (Except in Connection Pool Scenarios)
3			Rule 1.3: Ensuring a Started Transaction Is Committed or Rolled Back
4			Rule 1.4: Ensuring the Idle Timeout Duration Is Shorter Than SESSION_TIMEOUT Value When Connection Pool Is Used for Applications
5			Rule 1.5: Restoring Parameters to Default Values in Connections Before Returning Them to the Pool
6			Rule 1.6: Manually Clearing Temporary Tables Created with a Connection Before Returning it to the Pool
7	Obje ct desig n specif ication s	DATABAS E object design	Rule 2.1: Avoiding Direct Usage of Built-in Databases Such As postgres and gaussdb
8			Rule 2.2: Selecting the Suitable Database Code During Database Creation
9			Rule 2.3: Choosing the Right Database Type for Compatibility with the Database to Be Created
10			Suggestion 2.4: Creating the Objects with Associated Calculations in the Same Database
11		USER object design	Rule 2.5: Following the Least Privilege Principle and Avoiding Running Services Using Users with Special Permissions
12			Rule 2.6: Avoiding the Use of a Single Database Account for All Services
13		SCHEMA object design	Suggestion 2.7: Avoiding the Creation of Objects Under Other Users' Private Schemas
14		TABLESPA CE object design	Rule 2.8 Avoiding Tablespace Customization

N o.	Category		Rule/Suggestion
15		TABLE object design (prioritized)	Rule 2.9: Selecting the Optimal Distribution Method and Columns During Table Creation
16			Rule 2.10 Selecting an Optimal Storage Type During Table Creation
17			Rule 2.11 Selecting an Optimal Partitioning Policy During Table Creation
18			Suggestion 2.12: Designing Table Columns for Fast and Accurate Queries
19			Suggestion 2.13: Avoiding the Usage of Auto-increment Columns or Data Types
20		INDEX object design (prioritized)	Rule 2.14: Creating Necessary Indexes and Selecting Optimal Columns and Sequences for Them
21			Suggestion 2.15: Optimizing Performance by Choosing the Right Index Type and Avoiding Indexes for Column-Store Tables
22		VIEW object design	Suggestion 2.16: Limiting View Nesting to Three Layers
23	SQL development specifications	DDL operation specifications	Suggestion 3.1: Avoiding Performing DDL Operations (Except CREATE) During Peak Hours or in Long Transactions
24			Rule 3.2: Specifying the Scope of Objects to Be Deleted When Using DROP
25		INSERT operation specifications	Rule 3.3: Replacing INSERT with COPY for Efficient Multi-Value Batch Insertion
26			Suggestion 3.4: Avoiding Performing Real-time INSERT Operations on Common Column-store Tables
27		UPDATE/DELETE operation specifications	Suggestion 3.5: Preventing Simultaneous Updates or Deletions of the Same Row in a Row-store Table
28			Suggestion 3.6: Avoiding Frequent or Simultaneous UPDATE and DELETE Operations on Column-store Tables

N o.	Category		Rule/Suggestion
29		Specifications for the SELECT operation (including the query part in all syntaxes)	Rule 3.7: Avoiding Executing SQL Statements That Do Not Support Pushdown
30			Rule 3.8: Specifying Association Conditions when Multiple Tables Are Associated
31			Rule 3.9: Ensuring Consistency of Data Types in Associated Fields across Multiple Tables
32			Suggestion 3.10: Avoiding Function Calculation on Association and Filter Condition Fields
33			Suggestion 3.11: Performing Pressure Tests and Concurrency Control for Resource-intensive SQL Statements
34			Rule 3.12: Avoiding Excessive COUNT Operations on Large Row-store Tables
35			Suggestion 3.13: Avoid Getting Large Result Sets (Except for Data Exports)
36			Suggestion 3.14: Avoiding the Usage of SELECT * for Queries
37			Suggestion 3.15: Using WITH RECURSIVE with Defined Termination Condition for Recursion
38			Suggestion 3.16: Setting Schema Prefix for Table and Function Access
39			Suggestion 3.17: Identifying an SQL Statement with a Unique SQL Comment
40			Recommendation 3.18: Restricting SQL Statements to 64 KB in Length
41	Foreign table function development specifications	GDS foreign table	Rule 4.1 Deploying GDS on an Independent Server Outside the DWS Cluster
42		Foreign table for collaborative analysis	Rule 4.2 Avoiding Concurrent Access to Multiple Collaborative Analysis Foreign Tables Across Clusters

No.	Category	Rule/Suggestion
43	Stored procedure development specifications	Suggestion 5.1: Simplifying Stored Procedures and Avoiding Nesting
44		Rule 5.2: Avoiding Non-CREATE DDL Operations in Stored Procedures

3.2 DWS Connection Management Specifications

Rule 1.1: Configuring Load Balancing for DWS Clusters

NOTE

Impact of rule violation:

- Load imbalance causes performance problems and even service interruption.
- When a CN is faulty, services cannot be automatically recovered or the recovery may take a long time.

Solution:

- Configure ELB load balancing and connect the application to the load balancing IP address.

Rule 1.2: Ending the Database Connection After Necessary Operations (Except in Connection Pool Scenarios)

NOTE

Impact of rule violation:

- The number of idle connections exceeds the maximum limit, causing connection creation failure.
- Resource overload occurs because there are too many idle connections.

Solution:

- After the connection between the application and the database is established and used, manually end the connection.
- Set the **session_timeout** parameter on the service side to set the idle timeout duration. The connection will be automatically ended when the idle timeout duration expires.

Rule 1.3: Ensuring a Started Transaction Is Committed or Rolled Back

NOTE

Impact of rule violation:

- If a transaction remains uncommitted for an extended period, it blocks operations such as **ALTER**, thereby affecting all services.
- The number of idle connections exceeds the maximum limit, causing connection creation failure.

Solution:

- **autocommit** is enabled by default, so there is no need to manually commit any transaction unless you modify the default setting.
- If a transaction is explicitly started, it must be explicitly ended (either by committing or rolling back) once the relevant operations are finished.

Rule 1.4: Ensuring the Idle Timeout Duration Is Shorter Than **SESSION_TIMEOUT** Value When Connection Pool Is Used for Applications

NOTE

Impact of rule violation:

- The idle timeout mechanism on the service side clears connections in the connection pool, which negatively impacts connection reuse.

Solution:

- To ensure everything works correctly, make sure the idle timeout duration of the connection pool is shorter than the **SESSION_TIMEOUT** value in DWS. It is advised to adjust the idle timeout duration rather than modifying the **SESSION_TIMEOUT** value.

Rule 1.5: Restoring Parameters to Default Values in Connections Before Returning Them to the Pool

NOTE

Impact of rule violation:

- When a connection is reused by another service, the parameters set by the service may also be reused. This can result in performance issues or service errors.

Solution:

- Before returning the connection to the connection pool, use **SET SESSION AUTHORIZATION DEFAULT;RESET ALL;** to reset parameters.

Notes:

When connection pool is used for applications, if you set the global GUC parameter using **GS_GUC RELOAD** in DWS, restart the connection pool for the changes to be applied. This is because the modification only affects new connections in the connection pool.

Rule 1.6: Manually Clearing Temporary Tables Created with a Connection Before Returning it to the Pool

NOTE

Impact of rule violation:

- When a connection is reused by other services, an error may be reported when a temporary table is created.

Solution:

- Before returning a connection to the connection pool, use **DROP** to clear the temporary table created by the current session.

3.3 DWS Object Design Specifications

3.3.1 DATABASE Object Design

Rule 2.1: Avoiding Direct Usage of Built-in Databases Such As postgres and gaussdb

NOTE

Impact of rule violation:

- If the code or the compatibility setting of the built-in databases does not meet service requirements, you may need to migrate your data again.
- The time for changes to be applied may be prolonged if all services use built-in databases.

Solution:

- To meet the specific requirements of each service, it is recommended to create a dedicated database and allocate it accordingly.

Rule 2.2: Selecting the Suitable Database Code During Database Creation

NOTE

Impact of rule violation:

- Selecting the wrong database code may result in displaying garbled characters, and it is not possible to directly change the database code. In such cases, you will need to create a database and import the data again.

Solution:

- It is advisable to set the **ENCODING** to the UTF-8 format during database creation, unless there are specific requirements for a different encoding.

Rule 2.3: Choosing the Right Database Type for Compatibility with the Database to Be Created

NOTE

Impact of rule violation:

- Selecting the wrong type can lead to behavior inconsistencies after migrating the database from a different vendor to DWS. Unfortunately, it is not possible to directly change the compatibility setting. The only solution is to create a database and import the data again.

Solution:

- DWS supports compatibility with databases like Teradata, Oracle, and MySQL. You can specify **DBCOMPATIBILITY** to set the compatible database type when creating a database.

Suggestion 2.4: Creating the Objects with Associated Calculations in the Same Database

NOTE

Impact of rule violation:

- Cross-database access tends to have poorer performance compared to performing operations within the same database.

Solution:

- If multiple databases are created, it is advisable to create the objects requiring associated calculations in the same database.

3.3.2 USER Object Design

Rule 2.5: Following the Least Privilege Principle and Avoiding Running Services Using Users with Special Permissions

NOTE

Impact of rule violation:

- Administrators have full access to a lot of things in the system and using these users to run services can pose security and control risks.

Solution:

- It is advised to use common users for service running, reserving users with special permissions for management operations.

Rule 2.6: Avoiding the Use of a Single Database Account for All Services

NOTE

Impact of rule violation:

- Using a single database user for all services hinders effective service management and control. In abnormal situations, it becomes impossible to isolate specific users for emergency purposes.

Solution:

- Create administrators, service operation users, and O&M users for different purposes.
- Use different users to run different services for improved management and allocation of services and resources.

3.3.3 Schema Object Design

Suggestion 2.7: Avoiding the Creation of Objects Under Other Users' Private Schemas

 NOTE

A private schema refers to a schema with the same name as the user when the user is created. This schema is only accessible to the user.

Impact of rule violation:

- When you create an object under someone else's private schema, the permissions for that object are determined by the schema owner.

Solution:

- Create objects under your own private schema to have full control over the object permissions.

3.3.4 TABLESPACE Object Design

Rule 2.8 Avoiding Tablespace Customization

 NOTE

Impact of rule violation:

- In a distributed scenario, using a custom tablespace to create a table can result in the table data not being stored in a distributed manner by DN, leading to storage skew.

Solution:

- Use the built-in default tablespace when creating a table object.

3.3.5 TABLE Object Design (Prioritized)

Rule 2.9: Selecting the Optimal Distribution Method and Columns During Table Creation

 NOTE

Impact of rule violation:

- Incorrect distribution method and column selection can cause storage skew, deteriorate access performance, and even overload storage and computing resources.

Solution:

- When creating a table, it is important to use the **DISTRIBUTE BY** clause to explicitly specify the distribution method and distribution columns. The table below provides principles for selecting the distribution columns.

Table 3-2 Distribution column selection

Distribution Method	Description	Scenario
Hash	Table data is distributed to each DN based on the mapping between hash values generated by distribution columns and DNs. • Advantage: Each DN contains only part of data, which is space-saving. • Disadvantage: The even distribution of data depends heavily on the selection of distribution columns. If the join condition does not include the distribution columns of each node, data communication between nodes will be required.	Large tables and fact tables
RoundRobin	Table data is distributed to DNs in polling mode. • Advantage: Each DN only contains a portion of the data, taking up a small amount of space. Data is evenly distributed in polling mode and does not rely on distribution columns, eliminating data skews. • Disadvantage: Using distribution column conditions cannot eliminate or reduce inter-node communication. In this scenario, the performance is inferior to that of HASH.	Large tables, fact tables, and tables without proper distribution columns

Distribution Method	Description	Scenario
Replication	Full data in a table is copied to each DN in the cluster. • Advantage: Each DN holds the complete data of the table. The JOIN operation avoids data communication between nodes, reducing network overhead and the overhead of starting and stopping the STREAM thread. • Disadvantage: Each DN retains complete table data, which is redundant and occupies more storage space.	Small tables and dimension tables

Rule 2.10 Selecting an Optimal Storage Type During Table Creation

NOTE

Impact of rule violation:

- Row-store tables are not properly used. As a result, the query performance is poor and resources are overloaded.
- Improper use of column-store tables causes CU expansion, poor performance, and resource overload.

Solution:

- When creating a table, use the **orientation** parameter to explicitly specify the storage type. The following table describes the rules for selecting a storage type.

Table 3-3 Storage type selection

Storage Type	Advantageous Scenario	Disadvantageous Scenario
Row storage	• DML addition, deletion, and modification: scenarios with many UPDATE and DELETE operations • DML query: point query (simple index-based query that returns only a few records)	DML: statistical analysis query (with mass data involved in GROUP and JOIN processes) CAUTION When creating a row-store table (orientation is set to row), do not specify the compress attribute or use a row-store compressed table.

Storage Type	Advantageous Scenario	Disadvantageous Scenario
Column storage	• DML addition, deletion, and modification: INSERT batch import scenario (The number of data records imported to a single partition at a time is approximately 60,000 times the number of DNs or greater.) • DML query: statistical analysis query (with mass data involved in GROUP and JOIN processes)	• DML: scenarios with many UPDATE/DELETE operations or a small number of INSERT operations • DML query: high-concurrency point query
Hybrid row-column storage	DWS hybrid row-column storage are supported only by clusters of version 9.1.0 or later. This improves the real-time import capability and balances some advantages of row- and column-storage. • DML: INSERT/UPSERT/COPY and batch data import (except for the COPY micro-batch scenario, HStore Opt tables are better than HStore tables.) • DML: statistical analysis query (with mass data involved in GROUP and JOIN processes)	It is mainly used in extreme TP scenarios. • DML: high-frequency concurrent UPDATE/DELETE • DML: high-frequency concurrent point query

Rule 2.11 Selecting an Optimal Partitioning Policy During Table Creation

NOTE

Impact of rule violation:

Without partitioning, query performance and data governance efficiency will deteriorate. The larger the data volume, the greater the deterioration. The advantages of partitioning include:

- High query performance: The system queries only the relevant partitions rather than the whole table, improving the query efficiency.
- Improved data governance efficiency: In the data lifecycle management scenario, performing **TRUNCATE** or **DROP PARTITION** on historical partitions is much more efficient and effective than using **DELETE**.

Solution:

- Design partitions for tables that contain fields of the time type.

Table 3-4 Partitioning policy selection

Partitioning Policy	Description	Scenario
Range partitioning	Data is stored in different partitions based on the range of partition key values. The partition key ranges are consecutive but not overlapped.	1. The date or time field is used as the partition key. 2. Most queries contain partition keys as filter criteria. 3. Periodically delete data based on the partition key.
List partitioning	Partitioning is performed based on a unique list of partition key values.	1. A specific number of enumerated values are used as partition key values. 2. Most queries contain partition keys as filter criteria.

Suggestion 2.12: Designing Table Columns for Fast and Accurate Queries

NOTE

Impact of rule violation:

- The system may have limited storage space and low query efficiency.

Solution:

1. Design the table columns well for fast queries.
 - If you can select an integer, do not select the floating point or character type.
 - When using variable-length character type, specify the maximum length based on data features.
2. Design the table columns well for accurate queries.
 - Use the time type instead of the character type to store time data.
 - Use the minimum numeric type that meets the requirements. Avoid using bigint if int or smallint is sufficient to save space.
3. **Correctly use the constraints.**
 - Add **NOT NULL** constraints to columns that never have NULL values. The optimizer automatically optimizes the columns in certain scenarios.
 - Do not use the **DEFAULT** constraint for fields that can be supplemented at the service layer. Otherwise, unexpected results may be generated during data loading.
4. **Avoid unnecessary data type conversion.**
 - In tables that are logically related, columns having the same meaning should use the same data type.
 - Different types of comparison operations cause data type conversion, which may cause index and partition pruning failures and affect query performance.

Suggestion 2.13: Avoiding the Usage of Auto-increment Columns or Data Types

NOTE

Impact of rule violation:

- When auto-increment sequences or data types are heavily used, the GTM may become overloaded and slow down sequence generation.

Solution:

- Set a UUID to obtain a unique ID.
- If the auto-increment sequence must be used and there is no strict requirement for increasing order, you can set the cache, for example, **1000**, to reduce the pressure on GTM.

3.3.6 INDEX Object Design (Prioritized)

Rule 2.14: Creating Necessary Indexes and Selecting Optimal Columns and Sequences for Them

NOTE

Impact of rule violation:

- Redundant indexes consume unnecessary space and can impact data import efficiency.
- The column sequence in the composite index is incorrect, affecting the query efficiency.

Best practices:

The following conditions must be met when indexes are used:

- The index column should be a column commonly used for filtering or joining conditions.
- The index column should have more distinct values.
- When creating a multi-column combination index, prioritize columns with more distinct values.
- The number of indexes in a single table should be limited to less than five. You can control the number of indexes by combining them.
- In scenarios where data is added, deleted, or modified in batches, delete the index first and then add it back after the batch operation is complete to improve performance (real-time access may be affected).

Suggestion 2.15: Optimizing Performance by Choosing the Right Index Type and Avoiding Indexes for Column-Store Tables

NOTE

Impact of rule violation:

- Incorrect indexes do not improve column-store access and can negatively affect query performance.

Solution:

1. Specify the appropriate index type when creating indexes, avoiding the default PSORT index.
2. In point queries where small amounts of data need to be retrieved from mass datasets, consider using a B-tree index.
3. For high range query performance, create a partial cluster key (PCK) to quickly filter and scan fact tables using the min/max sparse index. Comply with the following rules to create a PCK:
 - [Notice] Only one PCK can be created in a table. A PCK can contain multiple columns, preferably no more than two columns.
 - [Suggestion] Create a PCK for the filter condition column of the expression (e.g., **col op const**, where **op** is the operator =, >, >=, <=, and <, and **const** is a constant value).

3.3.7 VIEW Object Design

Suggestion 2.16: Limiting View Nesting to Three Layers

NOTE

Impact of rule violation:

- Too many nested views can lead to unstable execution plans and unpredictable time consumption.
- The risk of rebuilding objects on which views depend is high and the probability of lock conflicts increases.

Solution:

- Create views based on physical tables.

3.4 DWS SQL Statement Development Specifications

3.4.1 DDL Operations

Suggestion 3.1: Avoiding Performing DDL Operations (Except CREATE) During Peak Hours or in Long Transactions

NOTE

Impact of rule violation:

DDL operations like **ALTER**, **DROP**, **TRUNCATE**, **REINDEX**, and **VACUUM FULL** have high lock levels and can block services during execution.

- During peak hours, these DDL operations with high lock levels should be avoided to prevent service blockage.
- Long transactions involving DDL operations with held or waited locks can also block services.

Solution:

- Choose off-peak hours or maintenance windows for DDL operations based on service periods. Specify the DDL execution environment and time consumption to avoid service blockage due to long lock waiting duration.

Rule 3.2: Specifying the Scope of Objects to Be Deleted When Using DROP

DANGER

Impact of rule violation:

Be cautious when using **DROP OBJECT** (e.g., **DATABASE**, **USER/ROLE**, **SCHEMA**, **TABLE**, **VIEW**) as it may cause data loss, especially with **CASCADE** deletions.

- **DROP DATABASE**: deletes all objects in the database.
- **DROP USER**: deletes the **USER** object and its schemas and table objects.
- **DROP SCHEMA**: deletes all objects in the schema.
- **DROP TABLE**: deletes the **TABLE** object and the indexes and views that depend on it.

Solution:

- Exercise caution when performing the **DROP** operation and back up data in advance.

3.4.2 INSERT Operation

Rule 3.3: Replacing INSERT with COPY for Efficient Multi-Value Batch Insertion

NOTE

Impact of rule violation:

- Parsing multiple values is time-consuming and resource-intensive, leading to low efficiency when importing data into the database.

Solution:

- Instead of using **INSERT VALUES**, the frontend should use APIs like CopyManager of JDBC.

Suggestion 3.4: Avoiding Performing Real-time INSERT Operations on Common Column-store Tables

NOTE

Impact of rule violation:

- Importing a small batch of data in real-time to a common column-store table can significantly expand the small CU, occupying a lot of storage space and impacting the query performance.

Solution:

- In real-time **INSERT** scenarios, evaluate the amount of data to be imported at once and the total amount of data. If the total amount of data is small, use row-store tables.
- In the real-time **INSERT** scenario, import around 60,000 data records to a single table, partition, or DN at a time. The minimum import batch is 5,000 data records.
- In the real-time **INSERT** scenario, use H-Store column-store tables (for clusters of version 8.3.0 or later).

3.4.3 UPDATE and DELETE Operations

Suggestion 3.5: Preventing Simultaneous Updates or Deletions of the Same Row in a Row-store Table

NOTE

Impact of rule violation:

- Concurrent **UPDATE** and **DELETE** operations on row-store tables may cause row lock blockage and distributed deadlocks, which can lead to service errors and performance degradation.

Solution:

- Group **UPDATE** and **DELETE** operations by primary key or distribution column. Perform parallel operations between groups while keeping operations within a group serial.

Suggestion 3.6: Avoiding Frequent or Simultaneous UPDATE and DELETE Operations on Column-store Tables

NOTE

Impact of rule violation:

- Frequent **UPDATE** and **DELETE** operations on column-store tables can result in CU bloat, leading to large space occupation and decreased access performance.
- Concurrent **UPDATE** and **DELETE** operations on row-store tables may cause row lock blockage and distributed deadlocks, which can lead to service errors and performance degradation.

Solution:

- Design tables with frequent **UPDATE** and **DELETE** operations as row-store tables.
- Group **UPDATE** and **DELETE** operations by primary key or distribution column. Perform parallel operations between groups while keeping operations within a group serial.

3.4.4 SELECT Operation

Rule 3.7: Avoiding Executing SQL Statements That Do Not Support Pushdown

NOTE

DWS uses a distributed architecture, and to achieve optimal performance, SQL statements need to be pushed down to utilize distributed compute resources.

Impact of rule violation:

- SQL statements that are not pushed down may experience poor execution performance and, in severe cases, can lead to CN resource bottlenecks, impacting overall services.

Solution:

- Do not use syntax or functions that cannot be executed near the data source. For details, see [Optimizing Statement Pushdown](#).

Rule 3.8: Specifying Association Conditions when Multiple Tables Are Associated

NOTE

Impact of rule violation:

- If no association condition is specified when linking multiple tables, it will result in a Cartesian product calculation. This can lead to an expanded result set, posing risks of performance issues and resource overload.

Solution:

- Specify filter and association conditions for each table during the association process.

Rule 3.9: Ensuring Consistency of Data Types in Associated Fields across Multiple Tables

NOTE

Impact of rule violation:

- Ensure consistent data types for associated fields to avoid unnecessary type conversions, data redistribution issues, and hindered generation of optimal plans.

Solution:

- Use the same data type for associated fields when tables are associated.

Suggestion 3.10: Avoiding Function Calculation on Association and Filter Condition Fields

NOTE

Impact of rule violation:

- In cases where function calculations are involved in association and filter conditions, the optimizer may fail to obtain accurate field statistics, impacting execution performance.

Solution:

- When comparing association condition fields, preprocess the data before importing it into the database, especially when calculations are required for comparison.
- When filter criteria are compared with constants, perform function calculation only on constant columns. The following is an example:

```
SELECT id, from_image_id, from_person_id, from_video_id
FROM face_data
WHERE SS.DEL_FLAG = 'N'
AND NVL(SS.DELETE_FLAG, 'N') = 'N'
The modification is as follows:
SELECT id, from_image_id, from_person_id, from_video_id
FROM face_data
where SS.DEL_FLAG = 'N'
AND (SS.DELETE_FLAG = 'N' or SS.DELETE_FLAG is null)
```

Suggestion 3.11: Performing Pressure Tests and Concurrency Control for Resource-intensive SQL Statements

NOTE

Impact of rule violation:

- Storage and computing resources are overloaded, and the overall running performance deteriorates.

Solution:

A resource-intensive SQL statement contains:

- A large number of **UNION ALL**.
- A large number of AGGs (such as **COUNT DISTINCT** and **MAX**).
- A lot of **JOIN** operations for a large number of tables.
- A large number of **STREAM** operators (plan dimension).

Before rolling out, conduct pressure tests and implement concurrency control for these SQL statements. If the resource capacity is exceeded, optimizing the service should be prioritized before reassessing the rollout plan.

Rule 3.12: Avoiding Excessive COUNT Operations on Large Row-store Tables

NOTE

If SSDs or other high-performance disk types are used, it may not be necessary to adhere strictly to this rule, but it is still crucial to monitor the I/O consumption.

Impact of rule violation:

- Performing frequent **COUNT** operations on large row-store tables can consume a significant amount of I/O resources, potentially leading to performance issues if an I/O bottleneck occurs.

Solution:

- Reduce the frequency of **COUNT** operations, use result caching, and collect statistics by partition to minimize I/O consumption.

Suggestion 3.13: Avoid Getting Large Result Sets (Except for Data Exports)

NOTE

Impact of rule violation:

- If you do not need to view all the results, querying ultra-large result sets becomes inefficient and wasteful in terms of resources.

Solution:

- Use the **LIMIT** clause to retrieve only the necessary result segments.
- Use a cursor to obtain the result sets by segment and set an appropriate value for **FETCH SIZE** if you need to query a large number of result sets.

Suggestion 3.14: Avoiding the Usage of **SELECT *** for Queries

NOTE

Impact of rule violation:

- Querying unnecessary columns increases the computing load and wastes computing resources.

Solution:

- Clearly list the fields required for the query in the **SELECT** statement to improve the query performance.

Suggestion 3.15: Using **WITH RECURSIVE** with Defined Termination Condition for Recursion

NOTE

Impact of rule violation:

- In cases where there is no specific termination condition, recursive operations can enter an infinite loop.
- Recursive operations generate duplicate data and occupy excessive resources.

Solution:

- Design proper termination conditions based on the volume and characteristics of the data in the service table.

Suggestion 3.16: Setting Schema Prefix for Table and Function Access

NOTE

Impact of rule violation:

- If the schema name prefix is not specified, the search will be performed sequentially across all tablespaces based on the tablespace list in the current **search_path**. This can lead to accessing unexpected tables due to schema switchover.

Solution:

- To enhance readability, stability, and portability, explicitly specify the schema prefix as **SCHEMA**. when accessing tables and function objects.

Suggestion 3.17: Identifying an SQL Statement with a Unique SQL Comment

NOTE

Impact of rule violation:

- The service's source tracing capability is limited. You can only verify it with R&D engineers using the database, user name, and client IP address.

Solution:

- You are advised to use **query_band**. The following is an example:
`SET query_band='JobName=abc;AppName=test;UserName=user';`
- Add a unique comment for each SQL statement to facilitate troubleshooting and application performance analysis. The following is an example of such comment.
`/* Module_name_Tool name_Job name_Step */`, for example, `/* mca_python_XXXXXX_step1 */ insert into xxx select ... from`

Recommendation 3.18: Restricting SQL Statements to 64 KB in Length

NOTE

Impact of rule violation:

- SQL parsing is time-consuming and difficult to maintain. Frequent execution of SQL statements leads to severe log expansion.

Solution:

- Set a 64 KB limit on SQL statements.

3.5 DWS Foreign Table Function Development Specifications

Rule 4.1 Deploying GDS on an Independent Server Outside the DWS Cluster

NOTE

Impact of rule violation:

If GDS is deployed in a DWS cluster, it contends for resources with CNs or DNPs in the cluster, leading to a decline in the performance of both GDS and CNs or DNPs.

Solution:

- Deploy GDS on an independent server outside the DWS cluster.
- Ensure that the disk capacity of the GDS server and the network bandwidth between the GDS server and the DWS cluster are planned according to the requirements.

Rule 4.2 Avoiding Concurrent Access to Multiple Collaborative Analysis Foreign Tables Across Clusters

NOTE

Principle description: When cluster A accesses data in cluster B through collaborative analysis, all DNs in cluster A establish connections and active sessions with the CNs in cluster B.

Impact of rule violation:

The CNs in cluster B are overloaded. As a result, the number of connections and active sessions exceeds the threshold, and the access is abnormal.

Solution:

Use foreign tables to access a single table instead of performing concurrent queries on multiple foreign tables. If it is not possible to avoid concurrent queries, calculate and limit the number of concurrent queries based on the number of DNs in cluster A and the normal service volume of cluster B. Additionally, increasing the values of **max_active_statements** and **max_connections** can help solve the problem.

3.6 DWS Stored Procedure Development Specifications

Suggestion 5.1: Simplifying Stored Procedures and Avoiding Nesting

NOTE

Impact of rule violation:

- The maintenance cost for complex and nested stored procedures is high, making fault locating and recovery time-consuming.

Solution:

- Avoid using stored procedures altogether or limit their usage to a single layer. Nested stored procedures should be avoided.
- Create a corresponding log table for the stored procedure design and record information before and after key steps in the log table. Follow the steps below to implement this.

Saving and viewing logs

Step 1 Create a log table.

```
CREATE TABLE func_exec_log
(
  id varchar2(32) default lower(sys_guid()),
  pro_name varchar2(60),
  exec_times int,
  log_date date,
  deal_date date,
  log_message text
);
```

Step 2 Create a table and import data.

```
CREATE TABLE demo_table(data_id int, data_number int);
INSERT INTO demo_table values(generate_series(1,1000),generate_series(1,1000));
```

Step 3 Create a service stored procedure.

```
CREATE OR REPLACE FUNCTION demo_table_process(out exe_info text)
LANGUAGE plpgsql
AS $$
declare v_count int;
pro_result text;
fun_name text;
```

```
exec_times int;
begin
fun_name := 'demo_table_process';
select nvl(max(exec_times), '0') + 1 into exec_times from func_exec_log where pro_name = fun_name;
-- Insert data into the service table.
insert into demo_table values (dbms_random.value(1, 1000)::int,generate_series(1,
dbms_random.value(10000, 20000)::int));
get diagnostics v_count = ROW_COUNT;
exe_info = sysdate || '# step1:insert count:' || v_count || ' rows';
-- Delete specified data from a service table.
delete from demo_table where data_id = dbms_random.value(1, 1000)::int;
get diagnostics v_count = ROW_COUNT;
exe_info = exe_info || sysdate || '# step2:delete count:' || v_count || ' rows';
-- Update service table data.
update demo_table set data_number = dbms_random.value(1, 100)::int where data_id =
dbms_random.value(1, 1000)::int;
exe_info = exe_info || sysdate || '# step3:update count:' || sql%rowcount || ' rows';
-- Record logs either before the entire program ends or after each step completes. You can also create a
function specifically for logging purposes.
insert into func_exec_log(pro_name, exec_times, log_date, deal_date, log_message) values
(fun_name,exec_times,sysdate,split_part(regexp_split_to_table(exe_info, ','), '#',
1),split_part(regexp_split_to_table(exe_info, ','), '#', 2));
-- EXCEPTION is used to ensure that logs can be properly recorded when the insertion, update, or deletion
exits abnormally.
EXCEPTION
WHEN OTHERS THEN
pro_result := exe_info || sysdate || '# exception error message is: ' || sqlerrm;
insert into func_exec_log(pro_name, exec_times, log_date, deal_date, log_message)
values(fun_name,exec_times,sysdate,split_part(regexp_split_to_table(pro_result, ','), '#',
1),split_part(regexp_split_to_table(pro_result, ','), '#', 2));
END; $$;
```

Step 4 Invoke the stored procedure (normal execution).

```
SELECT demo_table_process();
```

Step 5 View the created log table to check the service running status.

```
SELECT * FROM func_exec_log ORDER BY log_date desc,deal_date,log_message;
```

id	pro_name	exec_times	log_date	deal_date	log_message
637343d9f2f10ec605c7687ff700ffe	demo_table_process	1	2022-11-15 15:46:34	2022-11-15 15:46:11	step1:insert count:19125 rows
637343d9fe850e3105c8687ff700ffe	demo_table_process	1	2022-11-15 15:46:34	2022-11-15 15:46:33	step2:delete count:22 rows
637343d9068a0fd805c9687ff700ffe	demo_table_process	1	2022-11-15 15:46:34	2022-11-15 15:46:34	step3:update count:15 rows

Step 6 Invoke the stored procedure again to construct an execution exception.

```
SELECT demo_table_process(); -- Delete the data_number column of demo_table to construct an exception,
and then call the stored procedure again.
```

Step 7 View the log to check the service running status.

----End

Rule 5.2: Avoiding Non-CREATE DDL Operations in Stored Procedures

NOTE

Impact of rule violation:

- A stored procedure is a large transaction. If a non-CREATE DDL operation, especially one with a high lock level, is executed, it can block external access to related tables during the stored procedure's execution window.

Solution:

- Avoid using non-CREATE DDL operations within stored procedures whenever possible. If there is a necessity to use such operations, carefully assess the duration of the stored procedures and the potential impact of the DDL operations. It is advised to schedule non-CREATE DDL operations during off-peak hours when external access services are less active.

3.7 Detailed Design Rules for DWS Objects

3.7.1 DWS Database Object Naming Rules

The name of a database object must contain 1 to 63 characters, start with a letter or underscore (_), and can contain letters, digits, underscores (_), and dollar signs (\$). If the database uses GBK, UTF8, or SQL_ASCII, object names can include Chinese characters. With UTF8 or SQL_ASCII, each Chinese character counts as three, with a limit of 21 characters. With GBK, each Chinese character counts as two, with a limit of 31 characters. The **Latin1** character set does not support Chinese characters. The character set format is specified during database creation. For details, see "CREATE DATABASE" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

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

NOTE

You can use **SELECT * FROM pg_get_keywords ()** to obtain DWS keywords. For other ways to obtain the keywords, see Keywords in *Data Warehouse Service (DWS) SQL Syntax Reference*.

- [Proposal] Do not use strings enclosed in double quotation marks to define database object names. In DWS, double quotation marks specify that the enclosed database object names are case sensitive. Case sensitivity of database object names makes problem location difficult.
- [Proposal] Use the same naming format for database objects.
 - In a system undergoing incremental development or service migration, you are advised to comply with its historical naming conventions.
 - A database object name consists of letters, digits, and underscores (_); and cannot start with a digit. You are advised to use multiple words separated with hyphens (-).
 - You are advised to use intelligible names and common acronyms or abbreviations for database objects. Acronyms or abbreviations that are generally understood are recommended. For example, you can use English words indicating actual business terms. The naming format should be consistent within a cluster.

- A variable name must be descriptive and meaningful. It must have a prefix indicating its type.
- [Proposal] The name of a table object should indicate its main characteristics, for example, whether it is an ordinary, temporary, or unlogged table.
 - An ordinary table name should indicate the business relevant to a data set.
 - Temporary tables are named in the format of **tmp_Suffix**.
 - Unlogged tables are named in the format of **ul_Suffix**.
 - Foreign tables are named in the format of **f_Suffix**.

3.7.2 DWS Database Object Design Rules

3.7.2.1 DWS Database and Schema Design Rules

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

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

Database Design Suggestions

- Create databases as required. Do not use the default **gaussdb** database of a cluster.
- Create a maximum of three user-defined databases in a cluster.
- To make your database encoding compatible with most characters, you are advised to use the UTF-8 encoding when creating a database.
- Exercise caution when you set **ENCODING** and **DBCOMPATIBILITY** configuration items during database creation. In DWS, **DBCOMPATIBILITY** can be set to **TD**, **Oracle**, or **MySQL** to be compatible with Teradata, Oracle, or MySQL syntax, respectively. Syntax behavior may vary with the three modes. For details, see [Syntax Compatibility Differences Among Oracle, Teradata, and MySQL](#).
- By default, a database owner has all permissions for all objects in the database, including the deletion permission. Exercise caution when using the deletion permission.

Schema Design Suggestions

- To let a user access an object in a schema, grant the **usage** permission and the permissions for the object to the user, unless the user has the **sysadmin** permission or is the schema owner.
- To let a user create an object in the schema, grant the **CREATE** permission for the schema to the user.

- By default, a schema owner has all permissions for all objects in the schema, including the deletion permission. Exercise caution when using the deletion permission.

3.7.2.2 DWS Table Design Rules

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

- [Notice] Evenly distribute data on each DN to prevent data skew. If most data is stored on several DNs, the effective capacity of a cluster decreases. Select a proper distribution column to avoid data skew.
- [Notice] Evenly scan each DN when querying tables. Otherwise, DNs most frequently scanned will become the performance bottleneck. For example, when you use equivalent filter conditions on a fact table, the nodes are not evenly scanned.
- [Notice] Reduce the amount of data to be scanned. You can use the pruning mechanism of a partitioned table.
- [Notice] Minimize random I/O. By clustering or local clustering, you can sequentially store hot data, converting random I/O to sequential I/O to reduce the cost of I/O scanning.
- [Notice] Try to avoid data shuffling. To shuffle data is to physically transfer it from one node to another. This unnecessarily occupies many network resources. To reduce network pressure, locally process data, and to improve cluster performance and concurrency, you can minimize data shuffling by using proper association and grouping conditions.

Selecting a Storage Mode

[Proposal] Selecting a storage mode is the first step in defining a table. The storage mode mainly depends on the user's service type. For details, see [Table 3-5](#).

Table 3-5 Table storage modes and scenarios

Storage Mode	Application Scenarios
Row storage	• Point queries (simple index-based queries that only return a few records) • Scenarios requiring frequent addition, deletion, and modification
Column storage	• Statistical analysis queries (requiring a large number of association and grouping operations) • Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

When creating a table for analysis, make sure to set the **ORIENTATION** to column storage explicitly.

```
CREATE TABLE public.t1
(
  id integer not null,
  data integer,
  age integer
)
WITH (ORIENTATION =COLUMN);
```

Selecting a Distribution Mode

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

Table 3-6 Table distribution modes and scenarios

Distribution Mode	Description	Application Scenarios
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 a cluster.	Dimension tables and fact tables containing a small amount of data
Round-robin	Each row of the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN.	Fact tables that contain a large amount of data and cannot find a proper distribution column in hash mode

Selecting a Partitioning Mode

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

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

The example of a partitioned table definition is as follows:

```
CREATE TABLE staffS_p1
(
  staff_ID    NUMBER(6) not null,
  FIRST_NAME  VARCHAR2(20),
  LAST_NAME   VARCHAR2(25),
  EMAIL       VARCHAR2(25),
  PHONE_NUMBER VARCHAR2(20),
  HIRE_DATE   DATE,
  employment_ID VARCHAR2(10),
  SALARY      NUMBER(8,2),
  COMMISSION_PCT NUMBER(4,2),
  MANAGER_ID  NUMBER(6),
  section_ID  NUMBER(4)
)
PARTITION BY RANGE (HIRE_DATE)
```



```
(  
  PARTITION HIRE_19950501 VALUES LESS THAN ('1995-05-01 00:00:00'),  
  PARTITION HIRE_19950502 VALUES LESS THAN ('1995-05-02 00:00:00'),  
  PARTITION HIRE_maxvalue VALUES LESS THAN (MAXVALUE)  
);
```

Selecting a Distribution Key

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

- [Proposal] Select a column containing discrete data as the distribution key, so that data can be evenly distributed on each DN. If a single column is not discrete enough, consider using multiple columns as distribution keys. You can select the primary key of a table as the distribution key. For example, in an employee information table, select the certificate number column as the distribution key.
- [Proposal] If the first rule is met, do not select a column having constant filter conditions as the distribution key. For example, in a query on the **dwcjk** table, if the **zqdh** column contains the constant filter condition **zqdh='000001'**, avoid selecting the **zqdh** column as the distribution key.
- [Proposal] If the first and second rules are met, select the join conditions in a query as distribution keys. If a join condition is used as a distribution key, the data involved in a join task is locally distributed on DNs, which greatly reduces the data flow cost among DNs.

3.7.2.3 DWS Column Design Rules

Selecting a Data Type

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

- [Proposal] Use the most efficient data types allowed.
If all of the following number types provide the required service precision, they are recommended in descending order of priority: integer, floating point, and numeric.
- [Proposal] In tables that are logically related, columns having the same meaning should use the same data type.
- [Proposal] For string data, you are advised to use variable-length strings and specify the maximum length. To avoid truncation, ensure that the specified maximum length is greater than the maximum number of characters to be stored. You are not advised to use CHAR(n), BPCHAR(n), NCHAR(n), or CHARACTER(n), unless you know that the string length is fixed.

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

Common String Types

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

Table 3-7 Common string types

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

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

3.7.2.4 DWS Constraint Design Rules

DEFAULT and NULL Constraints

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

Partial Cluster Key

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

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

Unique Constraint

- [Notice] Both row-store and column-store tables support unique constraints.
- [Proposal] The constraint name should indicate that it is a unique constraint, for example, **UNI***Included columns*.

Primary Key Constraint

- [Notice] Both row-store and column-store tables support the primary key constraint.
- [Proposal] The constraint name should indicate that it is a primary key constraint, for example, **PK***Included columns*.

Check Constraint

- [Notice] Check constraints can be used in row-store tables but not in column-store tables.

- [Proposal] The constraint name should indicate that it is a check constraint, for example, **CKIncluded columns**.

3.7.2.5 Design Rules for DWS Views and Associated Tables

View Design

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

Joined Table Design

- [Proposal] Minimize joined columns across tables.
- [Proposal] Joined columns should use the same data type.
- [Proposal] The names of associated fields should show the associations. For example, they can use the same name.

3.7.3 DWS JDBC Configuration Rules

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

Connection Parameters

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

```
params = {  
  { "user", user },  
  { "database", database },  
  { "client_encoding", "UTF8" },  
  { "DateStyle", "ISO" },  
  { "extra_float_digits", "2" },  
  { "TimeZone", createPostgresTimeZone() },  
};
```

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

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

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

fetchsize

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

autocommit

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

Connection Releasing

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

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

[Proposal] Reset the session environment before releasing connections to the JDBC connection tool. Otherwise, historical session information may cause object conflicts.

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

CopyManager

[Proposal] In the scenario where the ETL tool is not used and real-time data import is required, it is recommended that you use the CopyManager interface driven by the DWS JDBC to import data in batches during application development.

For how to use CopyManager, see [CopyManager](#).

3.7.4 DWS SQL Writing Rules

DDL

- [Proposal] In DWS, you are advised to execute DDL operations, such as creating table or making comments, separately from batch processing jobs to avoid performance deterioration caused by many concurrent transactions.
- [Proposal] Execute data truncation after unlogged tables are used because DWS cannot ensure the security of unlogged tables in abnormal scenarios.
- [Proposal] Suggestions on the storage mode of temporary and unlogged tables are the same as those on base tables. Create temporary tables in the same storage mode as the base tables to avoid high computing costs caused by hybrid row and column correlation.
- [Proposal] The total length of an index column cannot exceed 50 bytes. Otherwise, the index size will increase greatly, resulting in large storage cost and low index performance.

- [Proposal] Do not delete objects using **DROP...CASCADE**, unless the dependency between objects is specified. Otherwise, the objects may be deleted by mistake.

Data Loading and Uninstalling

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

Type conversion

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

Query Operation

- [Proposal] Do not return a large number of result sets to a client except the ETL program. If a large result set is returned, consider modifying your service design.
- [Proposal] Perform DDL and DML operations encapsulated in transactions. Operations like table truncation, update, deletion, and dropping, cannot be rolled back once committed. You are advised to encapsulate such operations in transactions so that you can roll back the operations if necessary.
- [Proposal] During query compilation, you are advised to list all columns to be queried and avoid using *. Doing so reduces output lines, improves query performance, and avoids the impact of adding or deleting columns on front-end service compatibility.
- [Proposal] During table object access, add the schema prefix to the table object to avoid accessing an unexpected table due to schema switchover.
- [Proposal] The cost of joining more than eight tables or views, especially full joins, is difficult to be estimated. You are advised to use the **WITH TABLE AS** statement or other methods to create interim tables to improve the readability of SQL statements.
- [Proposal] Do not use Cartesian products or full joins. Cartesian products and full joins will result in a sharp expansion of result sets and poor performance.
- [Notice] Only **IS NULL** and **IS NOT NULL** can be used to determine NULL value comparison results. If any other method is used, NULL is returned. For example, **NULL** instead of expected Boolean values is returned for **NULL<>NULL**, **NULL=NULL**, and **NULL<>1**.

- [Notice] Do not use `count(col)` instead of `count(*)` to count the total number of records in a table. `count(*)` counts the NULL value (actual rows) while `count (col)` does not.
- [Notice] While executing `count(col)`, the number of NULL record rows is counted as 0. While executing `sum(col)`, NULL is returned if all records are NULL. If not all the records are NULL, the number of NULL record rows is counted as 0.
- [Notice] To count multiple columns using `count()`, column names must be enclosed with parentheses. For example, `count ((col1, col2, col3))`. Note: When multiple columns are used to count the number of NULL record rows, a row is counted even if all the selected columns are NULL. The result is the same as that when `count(*)` is executed.
- [Notice] Null records are not counted when `count(distinct col)` is used to calculate the number of non-null columns that are not repeated.
- [Notice] If all statistical columns are NULL when `count(distinct (col1,col2,...))` is used to count the number of unique values in multiple columns, Null records are also counted, and the records are considered the same.
- [Notice] When constants are used to filter data, the system searches for functions used for calculating these two data types based on the data types of the constants and matched columns. If no function is found, the system converts the data type implicitly. Then, the system searches for a function used for calculating the converted data type.

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

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

- [Proposal] Do not use scalar subquery statements. A scalar subquery appears in the output list of a **SELECT** statement. In the following example, the part enclosed in parentheses is a scalar subquery statement:

```
SELECT id, (SELECT COUNT(*) FROM films f WHERE f.did = s.id) FROM staffs_p1 s;
```

Scalar subqueries often result in query performance deterioration. During application development, scalar subqueries need to be converted into equivalent table associations based on the service logic.

- [Proposal] In **WHERE** clauses, the filtering conditions should be sorted. The condition that few records are selected for reading (the number of filtered records is small) is listed at the beginning.
- [Proposal] Filtering conditions in **WHERE** clauses should comply with unilateral rules. That is, when the column name is placed on one side of a comparison operator, the optimizer automatically performs pruning optimization in some scenarios. Filtering conditions in a **WHERE** clause will be displayed in **col op expression** format, where **col** indicates a table column, **op** indicates a comparison operator, such as = and >, and **expression** indicates an expression that does not contain a column name. For example:

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

The modification is as follows:

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

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

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

Convert the statement to the following:

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

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

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

Rewrite the statement as follows:

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

NOTE

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

3.7.5 Rules for Using Custom DWS External Functions (pgSQL/Java)

- [Notice] Java UDFs can perform some Java logic calculation. Do not encapsulate services in Java UDFs.

- [Notice] Do not connect to a database in any way (for example, by using JDBC) in Java functions.
- [Notice] Only the data types listed in the following table can be used. User-defined types and complex data types (Java Array and derived classes) are not supported.
- [Notice] User-defined aggregation functions (UDAFs) and user-defined table-generating functions (UDTFs) are not supported.

Table 3-8 PL/Java mapping for default data types

DWS	Java
BOOLEAN	boolean
"char"	byte
bytea	byte[]
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT4	float
FLOAT8	double
CHAR	java.lang.String
VARCHAR	java.lang.String
TEXT	java.lang.String
name	java.lang.String
DATE	java.sql.Timestamp
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

3.7.6 Rules for Using DWS PL/pgSQL

General Principles

1. Development shall strictly comply with design documents.
2. Program modules shall be highly cohesive and loosely coupled.
3. Proper, comprehensive troubleshooting measures shall be developed.

4. Code shall be reasonable and clear.
5. Program names shall comply with a unified naming rule.
6. Fully consider the program efficiency, including the program execution efficiency and database query and storage efficiency. Use efficient and effective processing methods.
7. Program comments shall be detailed, correct, and standard.
8. The commit or rollback operation shall be performed at the end of a stored procedure, unless otherwise required by applications.
9. Programs shall support 24/7 processing. In the case of an interruption, the applications shall provide secure, easy-to-use resuming features.
10. Application output shall be standard and simple. The output shall show the progress, error description, and execution results for application maintenance personnel, and provide clear and intuitive reports and documents for business personnel.

Programming Principles

1. Use bound variables in SQL statements in the PL/pgSQL.
2. **RETURNING** is recommended for SQL statements in PL/pgSQL.
3. Principles for using stored procedures:
 - a. Do not use more than 50 output parameters of the Varchar or Varchar2 type in a stored procedure.
 - b. Do not use the LONG type for input or output parameters.
 - c. Use the CLOB type for output strings that exceed 10 MB.
4. Variable declaration principles:
 - a. Use **%TYPE** to declare a variable that has the same meaning as that of a column or variable in an application table.
 - b. Use **%ROWTYPE** to declare a record that has the same meaning as that of a row in an application table.
 - c. Each line of a variable declaration shall contain only one statement.
 - d. Do not declare variables of the LONG type.
5. Principles for using cursors:
 - a. Explicit cursors shall be closed after being used.
 - b. A cursor variable shall be closed after being used. If the cursor variable needs to transfer data to an invoked application, the cursor shall be closed in the application. If the cursor variable is used only in a stored procedure, the cursor shall be closed explicitly.
 - c. Before using **DBMS_SQL.CLOSE_CURSOR** to close a cursor, use **DBMS_SQL.IS_OPEN** to check whether the cursor is open.
6. Principles for collections:
 - a. You are advised to use the **FOR ALL** statement instead of the **FOR** loop statement to reference elements in a collection.
7. Principles for using dynamic statements:
 - a. Dynamic SQL shall not be used in the transaction programs of online systems.

- b. Dynamic SQL statements can be used to implement DDL statements and system control commands in PL/pgSQL.
 - c. Variable binding is recommended.
- 8. Principles for assembling SQL statements:
 - a. You are advised to use bound variables to assemble SQL statements.
 - b. If the conditions for assembling SQL statements contain external input sources, the characters in the input conditions shall be checked to prevent attacks.
 - c. In a PL/pgSQL script, the length of a single line of code cannot exceed 2499 characters.
- 9. Principles for using triggers:
 - a. Triggers can be used to implement availability design in scenarios where differential data logs are irrelevant to service processing.
 - b. Do not use triggers to implement service processing functions.

Exception Handling Principles

Any error that occurs in a PL/pgSQL function aborts the execution of the function and related transactions. You can use a **BEGIN** block with an **EXCEPTION** clause to catch and fix errors.

- 1. In a PL/pgSQL block, if an SQL statement cannot return a definite result, you are advised to handle exceptions (if any) in **EXCEPTION**. Otherwise, unhandled errors may be transferred to the external block and cause program logic errors.
- 2. You can directly use the exceptions that have been defined in the system. DWS does not support custom exceptions.
- 3. A block containing an **EXCEPTION** clause is more expensive to enter and exit than a block without one. Therefore, do not use **EXCEPTION** without need.

Writing Standard

- 1. Variable naming rules:
 - a. The input parameter format of a procedure or function is **IN_Parameter_name**. The parameter name shall be in uppercase.
 - b. The output parameter format of a procedure or function is **OUT_Parameter_name**. The parameter name shall be in uppercase.
 - c. The format for input and output parameters in a procedure or function is **IO_Parameter name**, with the parameter name written in uppercase.
 - d. Variables used in procedures and functions shall be composed of **v_Variable_name**. The variable name shall be in lower case.
 - e. In query concatenation, the concatenation variable name of the **WHERE** statement shall be **v_where**, and the concatenation variable name of the **SELECT** statement shall be **v_select**.
 - f. The record type (TYPE) name shall consist of **T** and a variable name. The name shall be in uppercase.
 - g. A cursor name shall consist of **CUR** and a variable name. The name shall be in uppercase.

- h. The name of a reference cursor (REF CURSOR) shall consist of **REF** and a variable name. The name shall be in uppercase.
- 2. Rules for defining variable types:
 - a. Use **%TYPE** to declare the type of a variable that has the same meaning as that of a column in an application table.
 - b. Use **%ROWTYPE** to declare the type of a record that has the same meaning as that of a row in an application table.
- 3. Rules for writing comments:
 - a. Comments shall be meaningful and shall not just repeat the code content.
 - b. Comments shall be concise and easy to understand.
 - c. Comments shall be provided at the beginning of each stored procedure or function. The comments shall contain a brief function description, author, compilation date, program version number, and program change history. The format of the comments at the beginning of stored procedures shall be the same.
 - d. Comments shall be provided next to the input and output parameters to describe the meaning of variables.
 - e. Comments shall be provided at the beginning of each block or large branch to briefly describe the function of the block. If an algorithm is used, comments shall be provided to describe the purpose and result of the algorithm.
- 4. Variable declaration format:

Each line shall contain only one statement. To assign initial values, write them in the same line.
- 5. Letter case:

Use uppercase letters except for variable names.
- 6. Indentation:

In the statements used for creating a stored procedure, the keywords **CREATE**, **AS/IS**, **BEGIN**, and **END** at the same level shall have the same indent.
- 7. Statement rules:
 - a. For statements that define variables, Each line shall contain only one statement.
 - b. The keywords **IF**, **ELSE IF**, **ELSE**, and **END** at the same level shall have the same indent.
 - c. The keywords **CASE** and **END** shall have the same indent. The keywords **WHEN** and **ELSE** shall be indented.
 - d. The keywords **LOOP** and **END LOOP** at the same level shall have the same indent. Nested statements or statements at lower levels shall have more indent.

4 Creating and Managing DWS Database Objects

4.1 Creating and Managing DWS Databases

A database is a collection of objects such as tables, indexes, views, stored procedures, and operators. DWS supports the creation of multiple databases. However, a client program can connect to and access only one database at a time, and cross-database query is not supported.

Template and Default Databases

- DWS provides two template databases **template0** and **template1** and a default database **gaussdb**.
- By default, each newly created database is based on a template database. The DWS database uses **template1** by default. The encoding format is SQL_ASCII, and user-defined character encoding is not allowed. If you need to specify the character encoding when creating a database, use **template0** to create the database.
- Do not use a client or any other tools to connect to or to perform operations on both the two template databases.

NOTE

You can run the **show server_encoding** command to view the current database encoding.

Creating a Database.

Run the **CREATE DATABASE** statement to create a database.

```
CREATE DATABASE mydatabase;
```

 NOTE

- When you create a database, if the length of the database name exceeds 63 bytes, the server truncates the database name and retains the first 63 bytes. Therefore, you are advised to set the length of the database name to a value less than or equal to 63 bytes. Do not use multi-byte characters as object names. If an object whose name is truncated mistakenly cannot be deleted, delete the object using the name before the truncation, or manually delete it from the corresponding system catalog on each node.
- Database names must comply with the naming convention of SQL identifiers. The current user 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 different schemas in the same database.
- You must have the permission to create a database or the permission that the system administrator owns.

Viewing Databases

To view databases, perform the following steps:

- Run the `\l` meta-command to view the database list of the database system.
`\l`
- Querying the database list using the **pg_database** system catalog
`SELECT datname FROM pg_database;`

Modifying a Database

You can use the **ALTER DATABASE** statement modify database configuration such as the database owner, name, and default settings.

- Run the following command to set the default search path for the database:
`ALTER DATABASE mydatabase SET search_path TO pa_catalog,public;`
- Rename the database.
`ALTER DATABASE mydatabase RENAME TO newdatabase;`

Deleting a Database

You can run **DROP DATABASE** statement to delete a database. This statement deletes the system catalog of the database and the database directory on the disk. Only the database owner or system administrator can delete a database. A database being accessed by users cannot be deleted, You need to connect to another database before deleting this database.

Run the **DROP DATABASE** statement to delete a database:
`DROP DATABASE newdatabase;`

4.2 Creating and Managing DWS Schemas

A schema is the logical organization of objects and data in a database. Schema management allows multiple users to use the same database without interfering with each other. Third-party applications can be added to corresponding schemas to avoid conflicts.

The same database object name can be used in different schemas in a 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 in a database.

If a user is created, a schema named after the user will also be created in the current database.

Public mode

Each database has a schema named **public**. All users have the ability to use the public schema in the database, but only certain roles have the authority to create objects within it.

Creating a Schema

- Run the **CREATE SCHEMA** command to create a schema.

```
CREATE SCHEMA myschema;
```

To create or access an object in the schema, the object name in the command should be composed of the schema name and the object name, which are separated by a dot (.), for example, **myschema.table**.

- Users can create a schema owned by others. For example, run the following command to create a schema named **myschema** and set the owner of the schema to user **jack**:

```
CREATE SCHEMA myschema AUTHORIZATION jack;
```

If **authorization username** is not specified, the schema owner is the user who runs the command.

Modifying a Schema

- Run the **ALTER SCHEMA** command to change the schema name. Only the schema owner can change the schema name.

```
ALTER SCHEMA schema_name RENAME TO new_name;
```

- Run the **ALTER SCHEMA** command to change the schema owner.

```
ALTER SCHEMA schema_name OWNER TO new_owner;
```

Setting the Schema Search Path

The GUC parameter **search_path** specifies the schema search sequence. The parameter value is a series of schema names separated by commas (,). If no schema is specified during object creation, the object will be added to the first schema displayed in the search path. If there are objects with the same name in 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.

- Run the **SHOW** command to view the current search path.

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

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

- Run the **SET** command to modify the default schema of the current session. For example, if the search path is set to "**myschema, public**", **myschema** is searched first.

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

You can also run the **ALTER ROLE** command to set `search_path` for a role (user). For example:

```
ALTER ROLE jack SET search_path TO myschema, public;
```

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 a table **mytable** in **myschema**. Create a table in **schema_name.table_name** format.

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

- Query all data in the table **mytable** in **myschema**.

```
SELECT * FROM myschema.mytable;
```

```
id | name
```

```
----+-----
```

```
(0 rows)
```

Viewing a Schema

- Use the **current_schema()** function to view the current schema.

```
SELECT current_schema();
```

```
current_schema
```

```
-----
```

```
myschema
```

```
(1 row)
```

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

```
SELECT s.nspname,u.username AS nspowner FROM PG_NAMESPACE s, PG_USER u WHERE  
nspname='schema_name' AND s.nspowner = u.usesysid;
```

- To view a list of all schemas, query the system catalog **PG_NAMESPACE**.

```
SELECT * FROM PG_NAMESPACE;
```

- Use the **PGXC_TOTAL_SCHEMA_INFO** view to query the space usage of schemas in the cluster.

```
SELECT * FROM PGXC_TOTAL_SCHEMA_INFO;
```

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

```
SELECT distinct(tablename),schemaname FROM PG_TABLES where schemaname = 'pg_catalog';
```

Schema Permission Control

By default, a user can only access database objects in its own schema. To access objects in other schemas, the user must have the **usage** permission of the corresponding schema.

By granting the **CREATE** permission for a schema to a user, the user can create objects in this schema.

- Grant the **usage** permission of **myschema** to user **jack**.

```
GRANT USAGE ON schema myschema TO jack;
```


- Run the following command to revoke the **USAGE** permission for **myschema** from **jack**:
REVOKE USAGE ON schema myschema FROM jack;

Drop Schema

- Run the **DROP SCHEMA** command to delete an empty schema (no database objects in the schema).
DROP SCHEMA IF EXISTS myschema;
- By default, a schema must be empty before being deleted. To delete a schema and all its objects (such as tables, data, and functions), use the **CASCADE** keyword.
DROP SCHEMA myschema CASCADE;

System Schema

- Each database has a **pg_catalog** schema, which contains system catalogs and all built-in data types, functions, and operators. **pg_catalog** is a part of the search path and has the second highest search priority. It is searched after the schema of temporary tables and before other schemas specified in **search_path**. This search order ensures that database built-in objects can be found. To use a custom object that has the same name as a built-in object, you can specify the schema of the custom object.
- The **information_schema** consists of a collection of views that contain object information in a database. These views obtain system information from the system catalogs in a standardized way.

4.3 Creating and Managing DWS Tables

Creating a Table

You can run the **CREATE TABLE** command to create a table. When creating a table, you can define the following information:

- Columns and data type of the table.
- Table or column constraints that restrict a column or the data contained in a table. For details, see [Definition of Table Constraints](#).
- Distribution policy of a table, which determines how the DWS database divides data between segments. For details, see [Definition of Table Distribution](#).
- Table storage format. For details, see [Selecting a DWS Table Storage Model](#).
- Partition table information. For details, see [Creating and Managing DWS Partitioned Tables](#).

Example: Use **CREATE TABLE** to create a table **web_returns_p1**, use **wr_item_sk** as the distribution key, and sets the range distribution function through **wr_returned_date_sk**.

```
CREATE TABLE web_returns_p1
(
  wr_returned_date_sk integer,
  wr_returned_time_sk integer,
  wr_item_sk integer NOT NULL,
  wr_refunded_customer_sk integer
```

```
)  
WITH (orientation = column)  
DISTRIBUTE BY HASH (wr_item_sk)  
PARTITION BY RANGE(wr_returned_date_sk)  
(  
    PARTITION p2019 START(20191231) END(20221231) EVERY(10000),  
    PARTITION p0 END(maxvalue)  
);
```

Definition of Table Constraints

You can define constraints on columns and tables to restrict data in a table. However, there are the following restrictions:

- The primary key constraint and unique constraint in the table must contain a distribution column.
- Column-store tables support the **PARTIAL CLUSTER KEY** and table-level primary key and unique constraints, but do not support table-level foreign key constraints.
- Only the **NULL**, **NOT NULL**, and **DEFAULT** constant values can be used as column-store table column constraints.

Table 4-1 Table constraints

Constraint	Description	Example
Check constraint	A CHECK constraint allows you to specify that values in a specific column must satisfy a Boolean (true) expression.	Create the products table. The price column must be positive. CREATE TABLE products (product_no integer, name text, price numeric CHECK (price > 0));
NOT NULL constraint	A NOT NULL constraint specifies that a column cannot have null values. A non-null constraint is always written as a column constraint.	Create the products table. The values of product_no and name cannot be null. CREATE TABLE products (product_no integer NOT NULL, name text NOT NULL, price numeric);
UNIQUE constraint	A UNIQUE constraint specifies that the values in a column or a group of columns are all unique. If DISTRIBUTE BY REPLICATION is not specified, the column table that contains only unique values must contain distribution columns.	Create the products table. The values of product_no must be unique. CREATE TABLE products (product_no integer UNIQUE, name text, price numeric)DISTRIBUTE BY HASH(product_no);

Constraint	Description	Example
Primary key constraint	A primary key constraint is the combination of a UNIQUE constraint and a NOT NULL constraint. If DISTRIBUTE BY REPLICATION is not specified, the column set with a primary key constraint must contain distributed columns. If a table has a primary key, the column (or group of columns) of the primary key is selected as the distribution keys of the table by default.	Create the products table. The primary key constraint is product_no . <pre>CREATE TABLE products (product_no integer PRIMARY KEY, name text, price numeric)DISTRIBUTE BY HASH(product_no);</pre>
Partial cluster key	Partial cluster key can minimize or maximize sparse indexes to quickly filter base tables. Partial cluster key can specify multiple columns, but you are advised to specify no more than two columns.	Create the products table with PCK set to product_no : <pre>CREATE TABLE products (product_no integer, name text, price numeric, PARTIAL CLUSTER KEY(product_no)) WITH (ORIENTATION = COLUMN);</pre>

Definition of Table Distribution

DWS supports the following distribution modes: replication, hash, and roundrobin.

NOTE

The roundrobin distribution mode is supported only by cluster version 8.1.2 or later.

Policy	Description	Scenario	Advantages/Disadvantages
Replication	Full data in a table is stored on each DN in the cluster.	Small tables and dimension tables	• The advantage of replication is that each DN has full data of the table. During the join operation, data does not need to be redistributed, reducing network overheads and reducing plan segments (each plan segment starts a corresponding thread). • The disadvantage of replication is that each DN retains the complete data of the table, resulting in data redundancy. Generally, replication is only used for small dimension tables.
Hash	Table data is distributed on all DNs in the cluster.	Fact tables containing a large amount of data	• The I/O resources of each node can be used during data read/write, greatly improving the read/write speed of a table. • Generally, a large table (containing over 1 million records) is defined as a hash table.
Polling (Round-robin)	Each row in the table is sent to each DN in turn. Data can be evenly distributed on each DN.	Fact tables that contain a large amount of data and cannot find a proper distribution column in hash mode	• Round-robin can avoid data skew, improving the space utilization of the cluster. • Round-robin does not support local DN optimization like a hash table does, and the query performance of Round-robin is usually lower than that of a hash table. • If a proper distribution column can be found for a large table, use the hash distribution mode with better performance. Otherwise, define the table as a round-robin table.

Selecting a Distribution Key

If the hash distribution mode is used, a distribution key must be specified for the user table. When a record is inserted, the system hashes it based on the distribution key and then stores it on the corresponding DN.

Select a hash distribution key based on the following principles:

1. **The values of the distribution key should be discrete so that data can be evenly distributed on each DN.** You can select the primary key of the table as the distribution key. For example, for a person information table, choose the ID number column as the distribution key.
2. **Do not select the column that has a constant filter.** For example, if a constant constraint (for example, `zqdh= '000001'`) exists on the **zqdh** column in some queries on the **dwcjk** table, you are not advised to use **zqdh** as the distribution key.
3. **With the above principles met, you can select join conditions as distribution keys,** so that join tasks can be pushed down to DNs for execution, reducing the amount of data transferred between the DNs.

For a hash table, an inappropriate distribution key may cause data skew or poor I/O performance on certain DNs. Therefore, you need to check the table to ensure that data is evenly distributed on each DN. You can run the following SQL statements to check for data skew:

```
select
xc_node_id, count(1)
from tablename
group by xc_node_id
order by xc_node_id desc;
```

xc_node_id corresponds to a DN. Generally, **over 5% difference between the amount of data on different DNs is regarded as data skew. If the difference is over 10%, choose another distribution key.**

4. You are not advised to add a column as a distribution key, especially add a new column and use the SEQUENCE value to fill the column. (Sequences may cause performance bottlenecks and unnecessary maintenance costs.)

View the data in the table.

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

Deleting Data in a Table



CAUTION

Exercise caution when running the **DROP TABLE** and **TRUNCATE TABLE** statements. After a table is deleted, data cannot be restored.

- Delete the **customer_t1** table from the database.
DROP TABLE *customer_t1*;
- You can use **DELETE** or **TRUNCATE** to clear rows in a table without removing the definition of the table.

Delete all rows from the **customer_t1** table.

```
TRUNCATE TABLE customer_t1;
```

Delete all rows from the **customer_t1** table.

```
DELETE FROM customer_t1;
```

Delete all records whose **c_customer_sk** is **3869** from the **customer_t1** table.

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

Managing UNLOGGED Tables

UNLOGGED indicates an unlogged table. Unlogged tables are faster than regular tables because data written to them is not written to the WALs. However, an unlogged table is automatically cleared after a crash or unclean shutdown, incurring data loss risks. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are not automatically logged as well.

Usage scenario: Unlogged tables do not ensure safe data. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade. When creating an unlogged table, disable cnretry (that is, set the GUC parameter **max_query_retry_times** to 0).

Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should re-create the indexes with errors.

- Starting from version 9.1.0, UNLOGGED tables are automatically saved in the **pg_unlogged** tablespace and cannot be moved or assigned to other tablespaces.
- After an earlier version is upgraded to 9.1.0, the UNLOGGED table created in the earlier version is still stored in the original tablespace.

Version 9.1.0 has a script called **switch_unlogged_tablespace.py** that can move unlogged tables to optimize the recovery time objective (RTO). This script works together with the GUC parameter **enable_unlogged_tablespace_compat**.

1. The script is stored in the **\$GPHOME/script** directory. You can use the **-?** command to obtain help information.

```
[perfadm@linux83108 script]$ python3 switch_unlogged_tablespace.py -?
Usage:
  python3 switch_unlogged_tablespace.py -? | --help
  python3 switch_unlogged_tablespace.py -t query [--dbname=DBNAME] [--verbose]
  python3 switch_unlogged_tablespace.py -t switch [--dbname=DBNAME] [--without-disable]

General options:
  -t                Type of the command.
  -?, --help        Show help information for this utility, and exit the command line mode.

Options for query:
  --dbname          Database name.
  --verbose          List all unlogged tables.

Options for switch:
  --dbname          Database name.
  --without-disable Do not disable unlogged tablespace compatibility after switch all unlogged tables.
```

2. Migrate all unlogged tables (recommended).
`python3 switch_unlogged_tablespace.py -t switch`
3. After the migration, the GUC parameter **enable_unlogged_tablespace_compat** is automatically set to **off**.

NOTICE

After the upgrade to 9.1.0, you are advised to perform the following two steps to improve the instance restart RTO:

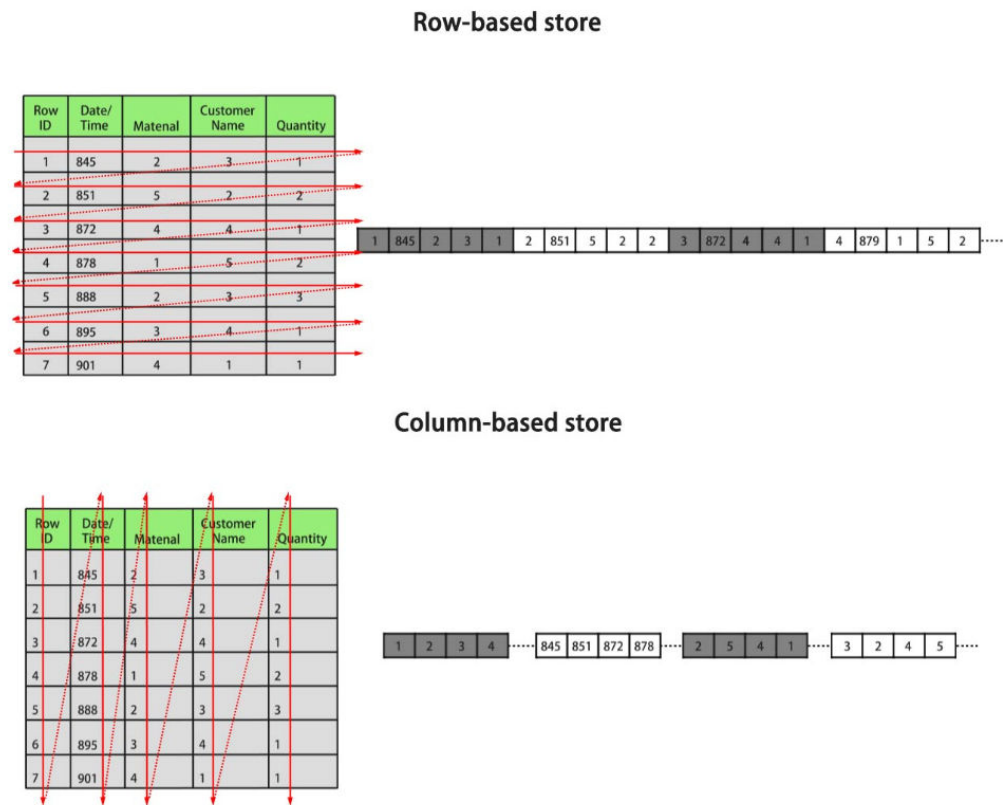
1. Use the **switch_unlogged_tablespace.py** script to migrate all unlogged tables to the **pg_unlogged** tablespace.
 2. If the old version does not use any unlogged table, you are advised to set the GUC parameter **enable_unlogged_tablespace_compat** to **OFF**.
-

4.4 Selecting a DWS Table Storage Model

DWS supports hybrid row and column storage. When creating a table, you can set the table storage mode to row storage or column storage.

Row storage stores tables to disk partitions by row, and column storage stores tables to disk partitions by column. By default, a table is created in row storage mode. For details about differences between row storage and column storage, see [Figure 4-1](#).

Figure 4-1 Differences between row storage and column storage



In the preceding figure, the upper left part is a row-store table, and the upper right part shows how the row-store table is stored on a disk; the lower left part is a column-store table, and the lower right part shows how the column-store table is stored on a disk.

The row/column storage of a table is specified by the **orientation** attribute in the table definition. The value **row** indicates a row-store table and **column** indicates a column-store table. The default value is **row**. Each storage mode applies to specific scenarios. Select an appropriate mode when creating a table.

Table 4-2 Table storage modes and scenarios

Storage Mode	Benefit	Drawback	Application Scenarios
Row storage	Data is stored by row. When you query a row of data, you can quickly locate the target row.	All data in the queried row is read while only a few columns are needed.	1. The number of columns in the table is small, and most fields in the table are queried. 2. Point queries (simple index-based query that returns only a few records) are performed. 3. Add, Delete, Modify, and Query operations on entire rows are frequently performed.
Column storage	1. Only necessary columns in a query are read. 2. The homogeneity of data within a column facilitates efficient compression.	It is not suitable for INSERT or UPDATE operations on a small amount of data.	1. Query a few columns in a table that contains a large number of columns. 2. Statistical analysis queries (requiring a large number of association and grouping operations) 3. Ad hoc queries (using uncertain query conditions and unable to utilize indexes to scan row-store tables)

Creating a Row-store Table

For example, to create a row-store table named **customer_t1**, run the following command:

```
CREATE TABLE customer_t1
(
  state_ID CHAR(2),
  state_NAME VARCHAR2(40),
  area_ID NUMBER
);
```

Creating a column-store table.

For example, to create a column-store table named **customer_t2**, run the following command:

```
CREATE TABLE customer_t2
(
  state_ID CHAR(2),
```

```
state_NAME VARCHAR2(40),  
area_ID NUMBER  
)  
WITH (ORIENTATION = COLUMN);
```

Table Compression

Table compression can be enabled when a table is created. Table compression enables data in the table to be stored in compressed format to reduce memory usage.

In scenarios where I/O is large (much data is read and written) and CPU is sufficient (little data is computed), select a high compression ratio. In scenarios where I/O is small and CPU is insufficient, select a low compression ratio. Based on this principle, you are advised to select different compression ratios and test and compare the results to select the optimal compression ratio as required. Specify a compressions ratio using the **COMPRESSION** parameter. The supported values are as follows:

- The valid value of column-store tables is **YES**, **NO**, **LOW**, **MIDDLE**, or **HIGH**, and the default value is **LOW**.
- The valid values of row-store tables are **YES** and **NO**, and the default is **NO**. (The row-store table compression function is not put into commercial use. To use this function, contact technical support.)

The service scenarios applicable to each compression level are described in the following table.

Compression Level	Application Scenario
LOW	The system CPU usage is high and the disk storage space is sufficient.
MIDDLE	The system CPU usage is moderate and the disk storage space is insufficient.
HIGH	The system CPU usage is low and the disk storage space is insufficient.

For example, to create a compressed column-store table named **customer_t3**, run the following command:

```
CREATE TABLE customer_t3  
(  
state_ID CHAR(2),  
state_NAME VARCHAR2(40),  
area_ID NUMBER  
)  
WITH (ORIENTATION = COLUMN, COMPRESSION=middle);
```

4.5 Creating and Managing DWS Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a

partition cable, and a physical piece is called a partition. Data is stored on these smaller physical pieces, namely, partitions, instead of the larger logical partitioned table. During conditional query, the system scans only the partitions that meet the conditions rather than scanning the entire table improving query performance.

For details about the syntax for maintaining partitioned tables, see "ALTER TABLE PARTITION" in *SQL Syntax Reference*.

Advantages of partitioned tables:

- Improved query performance. You can search in specific partitions, improving the search efficiency.
- Enhanced availability. If a partition is faulty, data in other partitions is still available.
- Improved maintainability. For expired historical data that needs to be periodically deleted, you can quickly delete it by dropping or truncate partitions.

Supported Table Partition Types

- Range partitioning: partitions are created based on a numeric range, for example, by date or price range.
- List partitioning: partitions are created based on a list of values, such as sales scope or product attribute. Only clusters of 8.1.3 and later versions support this function.

Choosing to Partition a Table

You can choose to partition a table when the table has the following characteristics:

- There are obvious ranges among the fields of the table.
A table is partitioned based on obvious rangeable fields. Generally, columns such as date, area, and value are used for partitioning. The time column is most commonly used.
- Queries to the table have obvious range characteristics.
If the queried data fall into specific ranges, its better tables are partitioned so that through partition pruning, only the queried partition needs to be scanned, improving data scanning efficiency and reducing the I/O overhead of data scanning.
- The table contains a large amount of data.
Scanning small tables does not take much time, therefore the performance benefits of partitioning are not significant. Therefore, you are advised to partition only large tables. In column-store table, each column is an independent file storage unit, and the minimum storage unit CU can store 60,000 rows of data. Therefore, for column-store partitioned tables, it is recommended that the data volume in each partition be greater than or equal to the number of DNs multiplied by 60,000.

Creating a Range Partitioned Table

Example: Create a table **web_returns_p1** partitioned by the range **wr_returned_date_sk**.

```
CREATE TABLE web_returns_p1
(
    wr_returned_date_sk    integer,
    wr_returned_time_sk    integer,
    wr_item_sk            integer NOT NULL,
    wr_refunded_customer_sk integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE (wr_returned_date_sk)
(
    PARTITION p2016 VALUES LESS THAN(20161231),
    PARTITION p2017 VALUES LESS THAN(20171231),
    PARTITION p2018 VALUES LESS THAN(20181231),
    PARTITION p2019 VALUES LESS THAN(20191231),
    PARTITION pxxxx VALUES LESS THAN(maxvalue)
);
```

Create partitions in batches, with fixed partition ranges. The following example can be used:

```
CREATE TABLE web_returns_p2
(
    wr_returned_date_sk    integer,
    wr_returned_time_sk    integer,
    wr_item_sk            integer NOT NULL,
    wr_refunded_customer_sk integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
    PARTITION p2016 START(20161231) END(20191231) EVERY(10000),
    PARTITION p0 END(maxvalue)
);
```

Partition the table **web_returns_p2** by date and time, using time as the partition key.

```
CREATE TABLE web_returns_p2
(
    id integer,
    idle numeric,
    IO numeric,
    scope text,
    IP text,
    time timestamp
)
WITH (TTL='7 days',PERIOD='1 day')
PARTITION BY RANGE(time)
(
    PARTITION P1 VALUES LESS THAN('2022-01-05 16:32:45'),
    PARTITION P2 VALUES LESS THAN('2022-01-06 16:56:12')
);
```

Creating a List Partitioned Table

A list partitioned table can use any column that allows value comparison as the partition key column. When creating a list partitioned table, you must declare the value partition for each partition.

Example: Create a list partitioned table **sales_info**.

```
CREATE TABLE sales_info
(
    sale_time timestamptz,
    period    int,
    city      text,
    price     numeric(10,2),
```

```
remark    varchar2(100)
)
DISTRIBUTE BY HASH(sale_time)
PARTITION BY LIST (period, city)
(
  PARTITION province1_202201 VALUES (('202201', 'city1'), ('202201', 'city2')),
  PARTITION province2_202201 VALUES (('202201', 'city3'), ('202201', 'city4'), ('202201', 'city5'))
);
```

Partitioning an Existing Table

A table can be partitioned only when it is created. If you want to partition a table, you must create a partitioned table, load the data in the original table to the partitioned table, delete the original table, and rename the partitioned table as the name of the original table. You must also re-grant permissions on the table to users. For example:

```
CREATE TABLE web_returns_p2
(
  wr_returned_date_sk    integer,
  wr_returned_time_sk    integer,
  wr_item_sk             integer NOT NULL,
  wr_refunded_customer_sk integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
  PARTITION p2016 START(20161231) END(20191231) EVERY(10000),
  PARTITION p0 END(maxvalue)
);
INSERT INTO web_returns_p2 SELECT * FROM web_returns_p1;
DROP TABLE web_returns_p1;
ALTER TABLE web_returns_p2 RENAME TO web_returns_p1;
GRANT ALL PRIVILEGES ON web_returns_p1 TO dbadmin;
GRANT SELECT ON web_returns_p1 TO jack;
```

Adding a Partition

Run the **ALTER TABLE** statement to add a partition to a partitioned table. For example, to add partition **P2020** to the **web_returns_p1** table, run the following command:

```
ALTER TABLE web_returns_p1 ADD PARTITION P2020 VALUES LESS THAN (20201231);
```

Add a partition to a list partitioned table. For example, add the **province3_202201** partition to the **sales_info** table.

```
ALTER TABLE sales_info ADD PARTITION province3_202201 VALUES (('202201', 'city6'), ('202201', 'city7'));
```

Splitting a Partition

The syntax for splitting a partition varies between a range partitioned table and a list partitioned table.

- Run the **ALTER TABLE** statement to split a partition in a range partitioned table. For example, the partition **pxxxx** of the table **web_returns_p1** is split into two partitions **p2020** and **p20xx** at the splitting point **20201231**.

```
ALTER TABLE web_returns_p1 SPLIT PARTITION pxxxx AT(20201231) INTO (PARTITION p2020,PARTITION p20xx);
```
- Run the **ALTER TABLE** statement to split a partition in a list partitioned table. For example, split the partition **province2_202201** of table **sales_inf** into two partitions **province3_202201** and **province4_202201**.

```
ALTER TABLE sales_info SPLIT PARTITION province2_202201 VALUES (('202201', 'city5')) INTO  
(PARTITION province3_202201, PARTITION province4_202201);
```

Merging Partitions

Run the **ALTER TABLE** statement to merge two partitions in a partitioned table. For example, merge partitions **p2016** and **p2017** of table **web_returns_p1** into one partition **p20162017**.

```
ALTER TABLE web_returns_p1 MERGE PARTITIONS p2016,p2017 INTO PARTITION p20162017;
```

Deleting a Partition

Run the **ALTER TABLE** statement to delete a partition from a partitioned table. For example, run the following command to delete partition **P2020** from the **web_returns_p1** table:

```
ALTER TABLE web_returns_p1 DROP PARTITION P2020;
```

Querying a Partition

- Query partition **p2019**.

```
SELECT * FROM web_returns_p1 PARTITION (p2019);  
SELECT * FROM web_returns_p1 PARTITION FOR (20201231);
```
- View partitioned tables using the system catalog **dba_tab_partitions**.

```
SELECT * FROM dba_tab_partitions where table_name='web_returns_p1';
```

Deleting a Partitioned Table

Run the **DROP TABLE** statement to delete a partitioned table.

```
DROP TABLE web_returns_p1;
```

Setting Whether a Partitioned Index Is Available

Create the local index **student_grade_index** for the partitioned table **customer_address** and set partition index names.

```
CREATE INDEX customer_address_index ON customer_address(ca_address_id) LOCAL  
(  
    PARTITION P1_index,  
    PARTITION P2_index,  
    PARTITION P3_index  
);
```

Rebuild all indexes on partition **P1** in the partitioned table **customer_address**.

```
ALTER TABLE customer_address MODIFY PARTITION P1 REBUILD UNUSABLE LOCAL INDEXES;
```

Set all indexes in partition **P3** of the partitioned table **customer_address** to be unusable.

```
ALTER TABLE customer_address MODIFY PARTITION P3 UNUSABLE LOCAL INDEXES;
```

Rebuilding Partition Indexes

For a partitioned table that has been running for a long time, indexes may generate fragments. Rebuilding indexes can improve query efficiency. The following describes how to rebuild indexes after data is inserted into partitions.

Step 1 Creates a partitioned table.

```
DROP TABLE IF EXISTS sales;
CREATE TABLE sales (
  id      INT,
  sale_date DATE,
  amount  DECIMAL(10,2)
)
PARTITION BY RANGE (sale_date)
(
  PARTITION p202310 VALUES LESS THAN ('2023-11-01'),
  PARTITION p202311 VALUES LESS THAN ('2023-12-01')
);
```

Step 2 Create an index.

```
CREATE INDEX idx_sale_date ON sales (sale_date) LOCAL;
```

Step 3 Insert partition data of October and November.

```
INSERT INTO sales PARTITION (p202310)
VALUES
(1, '2023-10-05', 100.50),
(2, '2023-10-10', 200.75);
INSERT INTO sales PARTITION (p202311)
VALUES
(3, '2023-11-15', 300.00);
```

Step 4 Rebuild indexes of the specific partition **p202310**.

```
ALTER TABLE sales REBUILD PARTITION p202310 WITHOUT UNUSABLE;
```

----End

Exchanging Partitions

The following example demonstrates how to migrate data from table **math_grade** to partition **math** in partitioned table **student_grade**.

1. Create the partitioned table **student_grade.**

```
CREATE TABLE student_grade (
  stu_name  char(5),
  stu_no    integer,
  grade     integer,
  subject   varchar(30)
)
PARTITION BY LIST(subject)
(
  PARTITION gym VALUES('gymnastics'),
  PARTITION phys VALUES('physics'),
  PARTITION history VALUES('history'),
  PARTITION math VALUES('math')
);
```

2. Add data to the partitioned table **student_grade.**

```
INSERT INTO student_grade VALUES
('Ann', 20220101, 75, 'gymnastics'),
('Jack', 20220103, 60, 'math'),
('Anna', 20220108, 56, 'history'),
('Jann', 20220107, 82, 'physics'),
('Molly', 20220104, 91, 'physics'),
('Sam', 20220105, 72, 'math');
```

3. Query the records of partition **math in **student_grade**.**

```
SELECT * FROM student_grade PARTITION (math);
```

The query result is as follows:

stu_name	stu_no	grade	subject
Jack	20220103	60	math

```
Sam | 20220105 | 72 | math
(2 rows)
```

4. Create an ordinary table **math_grade** that matches the definition of the partitioned table **student_grade**.

```
CREATE TABLE math_grade
(
    stu_name    char(5),
    stu_no     integer,
    grade       integer,
    subject     varchar(30)
);
```

5. Insert data to the **math_grade** table. The data in the **student_grade** partitioned table conforms to the partition rule of partition **math**.

```
INSERT INTO math_grade VALUES
('Ann', 20220101, 75, 'math'),
('Jack', 20220103, 60, 'math'),
('Anna', 20220108, 56, 'math'),
('Jann', 20220107, 82, 'math');
```

6. Migrate data from the ordinary table **math_grade** to partition **math** in the partitioned table **student_grade**.

```
ALTER TABLE student_grade EXCHANGE PARTITION (math) WITH TABLE math_grade;
```

7. Query the **student_grade** partitioned table. The results show that data in the **math_grade** table has exchanged with data in the **math** partition of the **student_grade** table.

```
SELECT * FROM student_grade PARTITION (math);
stu_name | stu_no | grade | subject
-----+-----+-----+-----
Anna    | 20220108 | 56 | math
Jack    | 20220103 | 60 | math
Ann     | 20220101 | 75 | math
Jann    | 20220107 | 82 | math
(4 rows)
```

8. Query the **math_grade** table. The result shows that the data stored in the **math** partition of the **student_grade** partitioned table has been exchanged to the **math_grade** table.

```
SELECT * FROM math_grade;
stu_name | stu_no | grade | subject
-----+-----+-----+-----
Jack    | 20220103 | 60 | math
Sam     | 20220105 | 72 | math
(2 rows)
```

4.6 Creating and Managing DWS Indexes

Indexes accelerate the data access speed but also add the processing time of the insert, update, and delete operations. Therefore, before creating an index, consider whether it is necessary and determine the columns where indexes will be created. You can determine whether to add an index for a table by analyzing the service processing and data use of applications, as well as columns that are frequently used as search criteria or need to be sorted.

Index type

- **B-tree:** The B-tree index uses a structure that is similar to the B+ tree structure to store data key values, facilitating index search. B-tree indexes support comparison and range queries.
- **GIN:** GIN indexes are reverse indexes and can process values that contain multiple keys (for example, arrays).

- **GiST:** GiST indexes are suitable for the set data type and multidimensional data types, such as geometric and geographic data types.
- **PSORT:** PSORT indexes are used to perform partial sort on column-store tables.

Row-based tables support B-tree (default), GIN, and GiST indexes. Column-based tables support PSORT (default), B-Tree, and GIN indexes.

NOTE

Create a B-tree index for point queries.

Index Selection Principles

Indexes are created based on columns in database tables. When creating indexes, you need to determine the columns, which can be:

- Columns that are frequently searched: The search efficiency can be improved.
- The uniqueness of the columns and the data sequence structures is ensured.
- Columns that usually function as foreign keys and are used for connections. Then the connection efficiency is improved.
- Columns that are usually searched for by a specified scope. These indexes have already been arranged in a sequence, and the specified scope is contiguous.
- Columns that need to be arranged in a sequence. These indexes have already been arranged in a sequence, so the sequence query time is accelerated.
- Columns that usually use the WHERE clause. Then the condition decision efficiency is increased.
- Fields that are frequently used after keywords, such as **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 sequenced scanning, the index will be used.
- After an index is successfully created, it must be synchronized with the associated table to ensure new data can be accurately located. Therefore, data operations increase. Therefore, delete unnecessary indexes periodically.

Creating an Index

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

NOTE

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

Table 4-3 Indexing Method

Indexing Method	Description
Unique index	Refers to an index that constrains the uniqueness of an index attribute or an attribute group. If a table declares unique constraints or primary keys, DWS automatically creates unique indexes (or composite indexes) for columns that form the primary keys or unique constraints. Currently, only B-tree can create a unique index in DWS.
Composite index	Refers to an index that can be defined for multiple attributes of a table. Currently, composite indexes can be created only for B-tree in DWS and a maximum of 32 columns can share a composite index.
Partial index	Refers to an index that can be created for subsets of a table. This indexing method contains only tuples that meet condition expressions.
Expression index	Refers to an index that is built on a function or an expression calculated based on one or more attributes of a table. An expression index works only when the queried expression is the same as the created expression.

- Run the following command to create an ordinary table:
CREATE TABLE *tpcds.customer_address_bak* **AS TABLE** *tpcds.customer_address*;

- Create a common index.

You need to query the following information in the **tpcds.customer_address_bak** table:

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

Generally, the database system needs to scan the **tpcds.customer_address_bak** table row by row to find all matched tuples. If the size of the **tpcds.customer_address_bak** table is large but only a few (possibly zero or one) of the WHERE conditions are met, the performance of this sequential scan is low. If the database system maintains an index on the **ca_address_sk** attribute, it can quickly locate the matching tuple by searching only a few layers in the search tree, significantly enhancing data query performance. Furthermore, indexes can improve the update and delete operation performance in the database.

Run the following command to create an index:

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

- Create a unique index.

If a table declares a unique constraint or primary key, DWS automatically creates a unique index (possibly a multi-column index) on the columns that form the primary key or unique constraint. If no unique constraint or primary key is specified during table creation, you can run the CREATE INDEX statement to create an index.

```
CREATE UNIQUE INDEX unique_index ON tpcds.customer_address_bak(ca_address_sk);
```

- Create a multi-column index.
Assume you need to frequently query records with **ca_address_sk** being **5050** and **ca_street_number** smaller than **1000** in the **tpcds.customer_address_bak** table. Run the following command:

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


Run the following command to define a multiple-column index on **ca_address_sk** and **ca_street_number** columns:

```
CREATE INDEX more_column_index ON tpcds.customer_address_bak(ca_address_sk,ca_street_number);
```
- Create a partition index.
If you only want to find records whose **ca_address_sk** is **5050**, you can create a partial index to facilitate your query.

```
CREATE INDEX part_index ON tpcds.customer_address_bak(ca_address_sk) WHERE ca_address_sk = 5050;
```
- Create an expression index.
Assume you need to frequently query records with **ca_street_number** smaller than **1000**, run the following command:

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


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

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

Querying an Index

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

```
SELECT RELNAME FROM PG_CLASS WHERE RELKIND='i';
```
- Run the following command to query information about a specified index:

```
\di+ index_wr_returned_date_sk
```

Recreating an Index

- Recreate the index **index_wr_returned_date_sk**.

```
REINDEX INDEX index_wr_returned_date_sk;
```
- Recreate all indexes of a table.

```
REINDEX TABLE tpcds.customer_address_bak;
```

Deleting an Index

You can use the **DROP INDEX** statement to delete indexes.

```
DROP INDEX index_wr_returned_date_sk;
```

4.7 Creating and Using DWS Sequences

A sequence is a database object that generates unique integers according to a certain rule and is usually used to generate primary key values.

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

- Set the data type of a column to sequence integer. A sequence will be automatically created by the database for this column.
- Use **CREATE SEQUENCE** to create a new sequence. Use the **nextval('sequence_name')** function to increment the sequence and return a

new value. Specify the default value of the column as the sequence value returned by the **nextval('sequence_name')** function. In this way, this column can be used as a unique identifier.

Creating a Sequence.

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

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

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

1. Create a sequence.

```
CREATE SEQUENCE seq1 cache 100;
```

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

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

NOTE

- Methods 1 and 2 are similar except that method 2 specifies cache for the sequence. A sequence using cache has holes (non-consecutive values, for example, 1, 4, 5) and cannot keep the order of the values. After a sequence is deleted, its sub-sequences will be deleted automatically. A sequence shared by multiple columns is not forbidden in a database, but you are not advised to do that.
- Currently, the preceding two methods cannot be used for existing tables.
- You are advised to control the number of sequences (including the SERIAL type). In distributed scenarios, sequence values are generated based on GTM. Frequent interaction with GTM severely affects performance. You are advised to reference the public sequence object or use UUIDs for unique values. To minimize the interaction with GTM, you are advised to set **CACHE** of a manually created sequence to a value greater than or equal to 10000.

Modifying a Sequence

The **ALTER SEQUENCE** statement changes the attributes of an existing sequence, including the owner, owning column, and maximum value.

- Associate the sequence with a column.

The sequence will be deleted when you delete the column or the table where the column resides.

```
ALTER SEQUENCE seq1 OWNED BY T2.id;
```

- Modify the maximum value of **serial** to **300**.

```
ALTER SEQUENCE seq1 MAXVALUE 300;
```

Deleting a Sequence

Run the **DROP SEQUENCE** command to delete a sequence. For example, to delete the sequence named **seq1**, run the following command:

```
DROP SEQUENCE seq1;
```

Precautions

Sequence values are generated by the GTM. By default, each request for a sequence value is sent to the GTM. The GTM calculates the result of the current value plus the step and then returns the result. As GTM is a globally unique node, generating default sequence numbers can cause performance issues. For operations that need frequent sequence number generation, such as bulkload data import, this is not recommended. For example, the **INSERT FROM SELECT** statement has poor performance in the following scenario:

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

To improve the performance, run the following statements (assume that data of 10,000 rows will be imported from *T1* to *newT1*):

```
INSERT INTO newT1(id, name) SELECT id,name from T1;  
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.

If BulkLoad is used, set sufficient cache for *newSeq1* and do not set **Maxvalue** or **Minvalue**. To improve the performance, database may push down the invocation of **nextval('sequence_name')** to DNs. 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 concurrent connection of BulkLoad to save the GTM connection resources. If the target table is a replication table (**DISTRIBUTE BY REPLICATION**), pushdown cannot be performed. If the data volume is large, this will be a disaster for the database. In addition, the database space may be exhausted. After the import is complete, do **VACUUM FULL**. Therefore, you are not advised to use sequences when BulkLoad is used.

After a sequence is created, a 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**.

4.8 Creating and Managing DWS Views

Views allow users to save queries. Views are not physically stored on disks. Queries to a view run as subqueries. A database only stores the definition of a view and does not store its data. The data is still stored in the original base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database. A view is triggered every time it is referenced.

Creating a view

Run the **CREATE VIEW** command to create a view.

```
CREATE OR REPLACE VIEW MyView AS SELECT * FROM tpcds.customer WHERE c_customer_sk < 150;
```

NOTE

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

View Details

- View the *MyView* view. Real-time data will be returned.

```
SELECT * FROM myview;
```
- Run the following command to query the views in the current user:

```
SELECT * FROM user_views;
```
- Run the following command to query all views:

```
SELECT * FROM dba_views;
```
- View details about a specified view.

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

```
\d+ dba_users
View "PG_CATALOG.DBA_USERS"
Column |      Type      | Modifiers | Storage | Description
-----+-----+-----+-----+-----
USERNAME | CHARACTER VARYING(64) |          | extended |
View definition:
SELECT PG_AUTHID.ROLNAME::CHARACTER VARYING(64) AS USERNAME
FROM PG_AUTHID;
```

Rebuilding a View

Run the **ALTER VIEW** command to rebuild a view without entering query statements.

```
ALTER VIEW myview REBUILD;
```

Deleting a View

Run the **DROP VIEW** command to delete a view.

```
DROP VIEW myview;
```

DROP VIEW ... The **CASCADE** command can be used to delete objects that depend on the view. For example, view A depends on view B. If view B is deleted, view A will also be deleted. Without the **CASCADE** option, the **DROP VIEW** command will fail.

4.9 Creating and Managing DWS Scheduled Tasks

DWS allows you to create scheduled tasks, which are automatically executed at specified time points, reducing O&M workload.

The database follows Oracle's scheduled task feature, allowing you to use the DBMS.JOB advanced package interfaces to create, run, delete, and modify scheduled tasks (including the task ID, task enabling and disabling, task triggering time, triggering interval, and task content).

NOTE

- The execution statements of scheduled tasks are not recorded in the [Real-time Top SQL](#) logs. The statements can be recorded only in versions later than 8.2.1.
- By default, DWS uses the UTC time. You need to convert the execution time of the scheduled task based on the time zone as required.

Periodic Task Management

Step 1 Creates a test table.

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

If the following information is displayed, the table has been created.

```
CREATE TABLE
```

Step 2 Create the customized storage procedure.

```
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 procedure has been created.

```
CREATE PROCEDURE
```

Step 3 Create a task.

- Create a task with unspecified **job_id** and execute the **PRC_JOB_1** storage procedure every two minutes.

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

- Create task with specified **job_id**.

```
call dbms_job.isubmit(2,'call public.prc_job_1(); ', sysdate, 'interval "1 minute"");  
isubmit  
-----  
(1 row)
```

Step 4 View the created task information about the current user in the **USER_JOBS** view.

Only the system administrator can access this system view. For details about the fields, see [Table 18-346](#).

```
select job,dbname,start_date,last_date,this_date,next_date,broken,status,interval,failures,what from
user_jobs;
job | dbname | start_date | last_date | this_date | next_date |
broken | status | interval | failures | what
-----+-----+-----+-----+-----+-----+-----
1 | db_demo | 2022-03-25 07:58:01.829436 | 2022-03-25 07:58:03.174817 | 2022-03-25 07:58:01.829436 |
2022-03-25 07:59:01 | n | s | interval '1 minute' | 0 | call public.prc
_job_1();
2 | db_demo | 2022-03-25 07:58:15.893383 | 2022-03-25 07:58:16.608959 | 2022-03-25 07:58:15.893383 |
2022-03-25 07:59:15 | n | s | interval '1 minute' | 0 | call public.prc
_job_1();
(2 rows)
```

Step 5 Stop a task.

```
call dbms_job.broken(1,true);
broken
-----
(1 row)
```

Step 6 Start a task.

```
call dbms_job.broken(1,false);
broken
-----
(1 row)
```

Step 7 Modify attributes of a task.

- Modify the **Next_date** parameter information about a task. For example, change the value of **Next_date** of Job1 to 1 hour.

```
call dbms_job.next_date(1, sysdate+1.0/24);
next_date
-----
(1 row)
```

- Modify the **Interval** parameter information of a task. For example, change the value of **Interval** of Job1 to 1 hour.

```
call dbms_job.interval(1,'sysdate + 1.0/24');
interval
-----
(1 row)
```

- Modify the **What** parameter information of a **JOB**. For example, change **What** of Job1 to insert into public.test values(333, sysdate+5).

```
call dbms_job.what(1,'insert into public.test values(333, sysdate+5);');
what
-----
(1 row)
```

- Modify **Next_date**, **Interval**, and **What** parameter information of **JOB**.

```
call dbms_job.change(1, 'call public.prc_job_1();', sysdate, 'interval "1 minute"');
change
-----
(1 row)
```

Step 8 Delete a job.

```
call dbms_job.remove(1);
remove
-----
(1 row)
```


Step 9 Set job permissions.

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

---End

4.10 Viewing DWS 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 DWS. You can collect information about the database by querying the system catalog.

Querying Database Tables

For example, query the **PG_TABLES** system catalog for all tables in the **public** schema.

```
SELECT distinct(tablename) FROM pg_tables WHERE SCHEMANAME = 'public';
```

Information similar to the following is displayed:

tablename
err_hr_staffs
test
err_hr_staffs_ft3
web_returns_p1
mig_seq_table
films4
(6 rows)

Viewing Database Users

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

```
SELECT * FROM pg_user;
username | usesysid | usecreatedb | usesuper | usecatupd | userepl | passwd | valbegin | valuntil | respool
| parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelim
it
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+
---
Ruby | 10 | t | t | t | t | ***** | | | default_pool | 0 |
dbadmin | 16393 | f | | f | f | f | ***** | | | default_pool | 0 |
lily | 16691 | f | f | f | f | ***** | | | default_pool | 0 |
jack | 70694 | f | f | f | f | ***** | | | default_pool | 0 |
(4 rows)
```

DWS uses Ruby to perform routine management and maintenance. You can add **WHERE usesysid > 10** to the **SELECT** statement to filter queries so that only specified user names are displayed.

```
SELECT * FROM pg_user WHERE usesysid > 10;
username | usesysid | usecreatedb | usesuper | usecatupd | userepl | passwd | valbegin | valuntil |
respool | parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelim
it
+-----+-----+-----+-----+-----+-----+-----+-----+
---
dbadmin | 16393 | f | | f | f | f | ***** | | | default_pool | 0 |
lily | 16691 | f | f | f | f | ***** | | | default_pool | 0 |
jack | 70694 | f | f | f | f | ***** | | | default_pool | 0 |
(3 rows)
```

Viewing and Stopping the Running Query Statements

You can view the running query statements in the [PG_STAT_ALL_INDEXES](#) view. Do as follows:

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

```
SET track_activities = on;
```

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

Step 2 View the running query statements. Run the following command to view the database names, users, query statuses, and PIDs of the running query statements:

```
SELECT datname, username, state, pid FROM pg_stat_activity;
```

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

To identify only active query statements, run the following command:

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

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

```
SELECT PG_TERMINATE_BACKEND(139834759993104);
```

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

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

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

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

 NOTE

If the **PG_TERMINATE_BACKEND** function is used to terminate the backend threads of the current session, the gsql client will be reconnected automatically rather than be logged out. The message "The connection to the server was lost." is returned. Attempting reset: Succeeded."

```
FATAL: terminating connection due to administrator command
FATAL: terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.
```

----End

5 Syntax Compatibility Differences Among Oracle, Teradata, and MySQL

DWS supports Oracle, Teradata, and MySQL compatibility types, each matching their respective syntax. Syntax behavior changes based on the selected type.

The database compatibility type can be specified during database creation (using the **DBCOMPATIBILITY** parameter). The following is an example of the syntax. For details, see "CREATE DATABASE".

```
CREATE DATABASE td_compatible_db DBCOMPATIBILITY 'TD'; --Create a Teradata-compatible database.
CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'ORA'; --Create an Oracle-compatible database.
CREATE DATABASE mysql_compatible_db DBCOMPATIBILITY 'MYSQL'; --Create a MySQL-compatible database.
```

Query results:

```
SELECT datname,datcompatibility FROM PG_DATABASE WHERE datname LIKE '%compatible_db';
  datname      | datcompatibility
-----+-----
td_compatible_db | TD
ora_compatible_db | ORA
mysql_compatible_db | MYSQL
(3 rows)
```

Table 5-1 Compatibility differences

Compatibility Item	Oracle	Teradata	MySQL
Data type: date	Converts the date data type to the timestamp data type which stores year, month, day, hour, minute, and second values.	Stores year, month, and date values.	Stores year, month, and date values.
Empty string	Only null is available.	Distinguishes empty strings from null values.	Distinguishes empty strings from null values.

Compatibility Item	Oracle	Teradata	MySQL
Conversion from an empty string to a number	Converts to null .	Converts to 0.	Converts to 0.
Automatic truncation of overlong characters	Not supported.	Supported (set GUC parameter td_compatible_truncation to ON).	Not supported.
VARCHAR + INT operation	Converts to BIGINT + INT calculation.	Converts to NUMERIC + NUMERIC calculation.	Converts to BIGINT + INT calculation.
Null concatenation	Returns a non-null object after combining a non-null object with null . For example, 'abc' null returns 'abc'.	The strict_text_concat_t option is added to the GUC parameter behavior_compat_options to be compatible with the Teradata behavior. After the null type is concatenated, null is returned. For example, 'abc' null returns null .	Compatible with MySQL behavior. After the null type is concatenated, null is returned. For example, 'abc' null returns null .

Compatibility Item	Oracle	Teradata	MySQL
Concatenation of the char(n) type	Removes spaces and placeholders on the right when the char(n) type is concatenated. For example, cast('a' as char(3)) 'b' returns 'ab'.	After the bpchar_text_without_rtrim option is added to the GUC parameter behavior_compat_options , when the char(n) type is concatenated, spaces are reserved and supplemented to the specified length <i>n</i> . Currently, ignoring spaces at the end of a string for comparison is not supported. If the concatenated string contains spaces at the end, the comparison is space-sensitive. For example, cast('a' as char(3)) 'b' returns 'a b'.	Removes spaces and placeholders on the right.
concat(str1, str2)	Returns the concatenation of all non-null strings.	Returns the concatenation of all non-null strings.	If an input parameter is null , null is returned.
Negative value processing in the left and right functions	Returns all characters except the first and last n characters.	Returns all characters except the first and last n characters.	Returns an empty string.

Compatibility Item	Oracle	Teradata	MySQL
lpad(string text, length int [, fill text]) rpad(string text, length int [, fill text])	Fills up the string to the specified length by appending the fill characters (a space by default). If the string is already longer than length then it is truncated (on the right). If fill is an empty string or length is a negative number, null is returned.	If fill is an empty string and the string length is less than the specified length , the original string is returned. If length is a negative number, an empty string is returned.	If fill is an empty string and the string length is less than the specified length , an empty string is returned. If length is a negative number, null is returned.
substr(str, s[, n])	If s is set to 0, the first n characters are returned.	If s is set to 0, the first n characters are returned.	If s is set to 0, an empty string is returned.
substring(str, s[, n]) substring(str [from s] [for n])	If s is set to 0, the first n - 1 characters are returned. If s is < 0, the first s + n - 1 characters are returned. If n is < 0, an error is reported.	If s is set to 0, the first n - 1 characters are returned. If s is < 0, the first s + n - 1 characters are returned. If n is < 0, an error is reported.	If s is set to 0, an empty string is returned. If s is < 0, n characters starting from the last s character are truncated. If n is < 0, an empty string is returned.
trim, ltrim, rtrim, btrim(string[, characters])	Removes the longest string that contains only the characters (a space by default) in the <i>characters</i> from a specified position of the <i>string</i> .	Removes the longest string that contains only the characters (a space by default) in the <i>characters</i> from a specified position of the <i>string</i> .	Removes the string that is equivalent to characters (a space by default) from a specified position of the <i>string</i> .
log(x)	Returns the logarithm with 10 as the base.	Returns the logarithm with 10 as the base.	Returns the natural logarithm.

Compatibility Item	Oracle	Teradata	MySQL
mod(x, 0)	Returns x if the divisor is 0 .	Returns x if the divisor is 0 .	Reports an error if the divisor is 0 .
to_char(date)	The maximum value of the input parameter can only be the maximum value of the timestamp type. The maximum value of the date type is not supported. The return value is of the timestamp type.	The maximum value of the input parameter can only be the maximum value of the timestamp type. The maximum value of the date type is not supported. The return value is of the date type in YYYY/MM/DD format. (The GUC parameter convert_empty_str_to_null_td is enabled.)	Only the timestamp type and the date type support the maximum input value. The return value is of the date type.
to_date, to_timestamp, and to_number processing of empty strings	Returns null .	Returns null . (The convert_empty_str_to_null_td parameter is enabled.)	to_date and to_timestamp returns null . If the parameter passed to to_number is an empty string, 0 is returned.
Return types of last_day and next_day	Returns values of the timestamp type.	Returns values of the timestamp type.	Returns values of the date type.
Return types of add_months	Returns values of the timestamp type.	Returns values of the timestamp type.	If the input parameter is of the date type, the return value is of the date type. If the input parameter is of the timestamp type, the return value is of the timestamp type. If the input parameter is of the timestamptz type, the return value is of the timestamptz type.

Compatibility Item	Oracle	Teradata	MySQL
CURRENT_TIME CURRENT_TIME(p)	Obtains the time of the current transaction. The return value is of the timetz type.	Obtains the time of the current transaction. The return value is of the timetz type.	Obtains the execution time of the current statement. The return value is of the time type.
CURRENT_TIMESTAMP CURRENT_TIMESTAMP(p)	Obtains the execution time of the current statement. The return value is of the timestampz type.	Obtains the execution time of the current statement. The return value is of the timestampz type.	Obtains the execution time of the current statement. The return value is of the timestamp type.
CURDATE	Not supported.	Not supported.	Obtains the execution date of the current statement. The return value is of the date type.
CURTIME(p)	Not supported.	Not supported.	Obtains the execution time of the current statement. The return value is of the time type.
LOCALTIME LOCALTIME(p)	Obtains the time of the current transaction. The return value is of the time type.	Obtains the time of the current transaction. The return value is of the time type.	Obtains the execution time of the current statement. The return value is of the timestamp type.
LOCALTIMESTAMP LOCALTIMESTAMP(p)	Obtains the time of the current transaction. The return value is of the timestamp type.	Obtains the time of the current transaction. The return value is of the timestamp type.	Obtains the execution time of the current statement. The return value is of the timestamp type.

Compatibility Item	Oracle	Teradata	MySQL
<code>SYSDATE</code> <code>SYSDATE(p)</code>	Obtains the execution time of the current statement. The return value is of the timestamp(0) type.	Obtains the execution time of the current statement. The return value is of the timestamp(0) type.	Obtains the current system time. The return value is of the timestamp(0) type. This function cannot be pushed down. You are advised to use <code>current_date</code> instead.
<code>now()</code>	Obtains the time of the current transaction. The return value is of the timestamptz type.	Obtains the time of the current transaction. The return value is of the timestamptz type.	Obtains the statement execution time. The return value is of the timestamptz type.
Operator '^'	Performs exponentiation.	Performs exponentiation.	Performs the XOR operation.
Expressions <code>GREATEST</code> and <code>LEAST</code>	Returns the comparison results of all non-null input parameters.	Returns the comparison results of all non-null input parameters.	If an input parameter is null , null is returned.
Different input parameter types of <code>CASE</code>, <code>COALESCE</code>, <code>IF</code>, and <code>IFNULL</code> expressions	Reports an error.	Is compatible with behavior of Teradata and supports type conversion between digits and strings. For example, if input parameters for <code>COALESCE</code> are of <code>INT</code> and <code>VARCHAR</code> types, the parameters are resolved as <code>VARCHAR</code> type.	Is compatible with behavior of MySQL and supports type conversion between strings and other types. For example, if input parameters for <code>COALESCE</code> are of <code>DATE</code> , <code>INT</code> , and <code>VARCHAR</code> types, the parameters are resolved as <code>VARCHAR</code> type.
Backquote (`)	Not supported.	Not supported.	Distinguishes MySQL reserved words from common characters.

Running SQL Statements

The following explains how to run SQL statements in a Teradata-compatible database. To see how Oracle and MySQL compatibility types behave differently, switch to either **ora_compatible_db** or **mysql_compatible_db**. Run these SQL

statements, replacing table names with **ora_table** or **mysql_table**, to observe the variations described in the preceding table.

```
CREATE TABLE td_table(a INT,b VARCHAR(5),c date);  
INSERT INTO td_table VALUES(1,null,CURRENT_DATE);  
INSERT INTO td_table VALUES(2,"",CURRENT_DATE);
```

Distinctions Between Empty Strings, NULL Values, and Date Displays

In both Teradata and MySQL compatibility types, empty strings and NULL values are distinct. However, in the Oracle compatibility type, they are considered the same. Additionally, dates convert to timestamps, displaying the year, month, day, hour, minute, and second.

Running SQL Statements in the Teradata Compatibility Type

```
SELECT a, b, b IS NULL AS null, c FROM td_table;
```

```
a | b | null | c  
---+---+-----+-----  
2 |  | f    | 2025-08-12  
1 |  | t    | 2025-08-12  
(2 rows)
```

```
SELECT CURRENT_DATE;
```

```
date  
-----  
2025-08-12  
(1 row)
```

Running SQL Statements in the Oracle Compatibility Type

```
SELECT a, b, b IS NULL AS null, c FROM ora_table;
```

```
a | b | null | c  
---+---+-----+-----  
1 |  | t    | 2025-08-12 00:00:00  
2 |  | t    | 2025-08-12 00:00:00  
(2 rows)
```

```
SELECT CURRENT_DATE;
```

```
date  
-----  
2025-08-12  
(1 row)
```

Running SQL Statements in the MySQL Compatibility Type

```
SELECT a, b, b IS NULL AS null, c FROM mysql_table;
```

```
a | b | null | c
---+---+---+-----
2 |  | f   | 2025-08-12
1 |  | t   | 2025-08-12
(2 rows)
```

```
SELECT CURRENT_DATE;
```

```
current_date
-----
2025-08-12
(1 row)
```

Processing Empty Strings

Different from the Oracle database, which processes an empty string as a NULL value, Teradata database converts an empty string to **0** by default. Therefore, when an empty string is queried, value **0** is found.

Similarly, in the Teradata compatibility type, the empty string is converted to **0** of the corresponding numeric type by default. In addition, '-', '+', and ' ' are converted to **0** by default in the Teradata compatibility type, but an error is reported for a decimal point string.

Processing Empty Strings in the Teradata Compatibility Type

```
SELECT b::int FROM td_table WHERE b = '';
```

```
b
--
0
(1 row)
```

Processing Empty Strings in the Oracle Compatibility Type

```
SELECT b::int FROM ora_table WHERE b = '';
```

```
b
--
(0 rows)
```

Processing Empty Strings in the MySQL Compatibility Type

```
SELECT b::int FROM mysql_table WHERE b = '';
```

```
b
--
0
(1 row)
```

Automatic Truncation of Overlong Characters

- In the Teradata compatibility type, if **td_compatible_truncation** is set to **on**, a long character string will be automatically truncated. If later **INSERT** statements (not involving foreign tables) insert long strings to columns of char- and varchar-typed columns in the target table, the system will truncate the long strings to ensure no strings exceed the maximum length defined in the target table.
- In the Oracle and MySQL compatibility types, an error is reported when an overlong string is inserted.

Automatic Character Truncation in the Teradata Compatibility Type

```
SHOW td_compatible_truncation;  
SET td_compatible_truncation = ON;  
INSERT INTO td_table VALUES(3,'12345678',CURRENT_DATE);  
SELECT * FROM td_table WHERE a = 3;
```

a	b	c
3	12345	2025-08-12

(1 row)

Automatic Character Truncation in the Oracle Compatibility Type

The **td_compatible_truncation** parameter is invalid. If an overlong character is inserted, an error is reported.

```
SHOW td_compatible_truncation;  
SET td_compatible_truncation = ON;  
INSERT INTO ora_table VALUES(3,'12345678',CURRENT_DATE);
```

```
ERROR:  dn_6001_6002: value too long for type character varying(5)  
CONTEXT:  referenced column: b
```

Automatic Character Truncation in the MySQL Compatibility Type

The **td_compatible_truncation** parameter is invalid. If an overlong character is inserted, an error is reported.

```
SHOW td_compatible_truncation;  
SET td_compatible_truncation = ON;  
INSERT INTO mysql_table VALUES(3,'12345678',CURRENT_DATE);
```

```
ERROR:  dn_6001_6002: value too long for type character varying(5)  
CONTEXT:  referenced column: b
```

Converting VARCHAR + INT Calculation to NUMERIC + NUMERIC Calculation

- In the Teradata compatibility type, the VARCHAR + INT calculation is converted to the NUMERIC + NUMERIC calculation.
- In the Oracle and MySQL compatibility types, BIGINT + INT calculation is used.

Calculation Conversion in the Teradata Compatibility Type

```
EXPLAIN VERBOSE SELECT b + a FROM td_table WHERE a = 3;
```

```
QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+
1 | -> Data Node Scan on "__REMOTE_FQS_QUERY__" |      0 |           |         | 0.00

Targetlist Information (identified by plan id)
-----
1 --Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: (((td_table.b)::numeric + (td_table.a)::numeric))
   Node/s: All datanodes (v3_logical, bucket:24)
   Remote query: SELECT b::numeric + a::numeric AS "?column?" FROM public.td_table WHERE a = 3
```

Calculation Conversion in the Oracle Compatibility Type

```
INSERT INTO ora_table VALUES(3,'12345',CURRENT_DATE);
EXPLAIN VERBOSE SELECT b + a FROM ora_table WHERE a = 3;
```

```
QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+
1 | -> Data Node Scan on "__REMOTE_FQS_QUERY__" |      0 |           |         | 0.00

Targetlist Information (identified by plan id)
-----
1 --Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: (((ora_table.b)::bigint + ora_table.a))
   Node/s: All datanodes (v3_logical, bucket:24)
   Remote query: SELECT b::bigint + a AS "?column?" FROM public.ora_table WHERE a = 3
```

Calculation Conversion in the MySQL Compatibility Type

```
INSERT INTO mysql_table VALUES(3,'12345',CURRENT_DATE);
EXPLAIN VERBOSE SELECT b + a FROM mysql_table WHERE a = 3;
```

```
QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+
1 | -> Data Node Scan on "__REMOTE_FQS_QUERY__" |      0 |           |         | 0.00

Targetlist Information (identified by plan id)
-----
1 --Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: (((mysql_table.b)::bigint + mysql_table.a))
   Node/s: All datanodes (v3_logical, bucket:24)
   Remote query: SELECT b::bigint + a AS "?column?" FROM public.mysql_table WHERE a = 3
```

Concatenating a Null Value

- Teradata compatibility type: The **strict_text_concat_td** option is added for GUC parameter **behavior_compat_options**. Concatenating the date, time, number, string, and **null** will return **null**.
- Oracle compatibility type: Concatenating a non-null object with NULL values returns a non-null object. For example, 'abc'||null returns 'abc'.

- MySQL compatibility type: Concatenating null types returns **null**, matching MySQL's behavior. For example, 'abc' || null returns **null**.

Concatenating a Null Value in the Teradata Compatibility Type

```
SET behavior_compat_options = 'strict_text_concat_td';  
SELECT '2024-02-07 12:12:12'::TIMESTAMP || NULL;  
SELECT '12:12:12'::TIME || NULL;  
SELECT '12'::TINYINT || NULL;  
SELECT 'abc'::CHAR(10) || NULL;
```

```
?column?  
-----  
  
(1 row)
```

Concatenating a Null Value in the Oracle Compatibility Type

```
SELECT 'abc'::CHAR(10) || NULL;
```

```
?column?  
-----  
abc  
(1 row)
```

Concatenating a Null Value in the MySQL Compatibility Type

```
SELECT 'abc'::CHAR(10) || NULL;
```

```
?column?  
-----  
  
(1 row)
```

Concatenating the char(n) Type

- In the Teradata compatibility type, the **bpchar_text_without_rtrim** option is added to GUC parameter **behavior_compat_options**. When the char(n) type is concatenated, spaces are reserved and padded to the specified length *n*.
- In the Oracle and MySQL compatibility types, spaces are not reserved.

Concatenating the char(n) Type in the Teradata Compatibility Type

If **a** has three spaces and converts to char(10), it fills up to 10 characters since its length is under 10. This uses 10 bytes of storage, with each byte holding 8 bits, totaling 80 bits.

```
SET behavior_compat_options = 'bpchar_text_without_rtrim';  
SELECT bit_length('a '::char(10));
```

```
bit_length
-----
80
(1 row)
```

Concatenating the char(n) Type in the Oracle and MySQL Compatibility Types

In the Oracle and MySQL compatibility types, no extra space is reserved. The letter **a** uses just one character and takes up one byte of storage. As a result, the output is **8**.

```
SELECT bit_length('a '::char(10));
```

```
bit_length
-----
8
(1 row)
```

Using the CONCAT Function

In the Teradata and Oracle compatibility types, **concat(str1,str2)** returns all non-null character strings. In the MySQL compatibility type, it returns **null** if an input parameter contains **null**.

Using the CONCAT Function in the Teradata and Oracle Compatibility Types

```
SELECT concat(null, 'World!');
```

```
concat
-----
World!
(1 row)
```

Using the CONCAT Function in the MySQL Compatibility Type

```
SELECT concat(null, 'World!');
```

```
concat
-----
(1 row)
```

Negative Value Processing in the left and right Functions

In Teradata and Oracle compatibility types, the left function removes the last **|n|** characters, while the right function removes the first **|n|** characters. In the MySQL compatibility type, an empty string is returned.

Negative Value Processing in the Teradata and Oracle Compatibility Types

```
SELECT left('abcde', -2);
```

```
left
-----
abc
(1 row)
```

```
SELECT right('abcde', -2);
```

```
right
-----
cde
(1 row)
```

Negative Value Processing in the MySQL Compatibility Type

```
SELECT left('abcde', -2);
```

```
left
-----
(1 row)
```

```
SELECT right('abcde', -2);
```

```
right
-----
(1 row)
```

Empty String Processing in the lpad and rpad Functions

- In the Teradata compatibility type, when **fill** is empty and the input string's length is shorter than **length**, both lpad and rpad functions output the original string unchanged. If the value of **length** is negative, an empty string is returned.
- In the Oracle compatibility type, these functions also return **null**.
- In the MySQL compatibility type, if **fill** is an empty string and the string length is less than the specified **length**, an empty string is returned. If **length** is a negative number, **null** is returned.

Empty String Processing in the Teradata Compatibility Type (lpad and rpad)

```
SELECT lpad('a',7,'');
```

```
lpad
-----
a
(1 row)
```

```
SELECT lpad('a',-7,"");
```

```
lpad
-----
(1 row)
```

```
SELECT rpad('a',7,"");
```

```
rpad
-----
a
(1 row)
```

```
SELECT rpad('a',-7,"");
```

```
rpad
-----
(1 row)
```

Empty String Processing in the Oracle Compatibility Type (lpad and rpad)

In the Oracle compatibility type, the lpad function returns **null**. The rpad function handles empty strings similarly; no further details are provided here.

```
SELECT lpad('a',7,"");
```

```
lpad
-----
(1 row)
```

```
SELECT lpad('a',-7,"");
```

```
lpad
-----
(1 row)
```

Empty String Processing in the MySQL Compatibility Type (lpad and rpad)

```
SELECT lpad('a',7,"");
```

```
lpad
-----
(1 row)
```

```
SELECT lpad('a',-7,"");
```

```
lpad
-----
(1 row)
```

Using the SUBSTR Function

In the Teradata and Oracle compatibility types, the `substr(str, s[, n])` function gives the first *n* characters if *s* is 0. In the MySQL compatibility type, it returns an empty string.

Using the SUBSTR Function in the Teradata and Oracle Compatibility Types

```
SELECT SUBSTR('Hello, World!', 0, 5);
```

```
substr
-----
Hello
(1 row)
```

Using the SUBSTR Function in the MySQL Compatibility Type

```
SELECT SUBSTR('Hello, World!', 0, 5);
```

```
substr
-----
(1 row)
```

Using the SUBSTRING Function

The `substring(str [from s] [for n])` function works differently based on its inputs in the Teradata-compatible and Oracle-compatible modes:

- If *s* is set to 0, the first *n* - 1 characters are returned.
- If *s* is less than 0, the first *s* + *n* - 1 characters are returned.
- If *n* is less than 0, an error is reported.

In the MySQL compatibility type:

- If *s* is 0, an empty string is returned.
- If *s* is less than 0, characters starting from the last *|s|* character are truncated.
- If *n* is less than 0, an empty string is returned.

Using the SUBSTRING Function in the Teradata and Oracle Compatibility Types

Example 1: If *s* is 0 and *n* is 5, the first four characters of **Hello, World!** are returned.

```
SELECT SUBSTRING('Hello, World!' FROM 0 FOR 5);
```

```
substring
-----
Hell
(1 row)
```

Example 2: If **s** is **-1** and **n** is **4**, the first two characters of "Hello, World!" are returned.

```
SELECT SUBSTRING('Hello, World!' FROM -1 FOR 4);
```

```
substring
-----
He
(1 row)
```

Example 3: If **n** is **-1**, an error is reported.

```
SELECT SUBSTRING('Hello, World!' FROM -1 FOR -1);
```

```
ERROR:  negative substring length not allowed
CONTEXT:  referenced column: substring
```

Using the SUBSTRING Function in the MySQL Compatibility Type

Example 1: An empty string is returned when **s** is **0** and **n** is **5**.

```
SELECT SUBSTRING('Hello, World!' FROM 0 FOR 5);
```

```
substring
-----
 
(1 row)
```

Example 2: If **s** is **-1** and **n** is **4**, the last character is returned.

```
SELECT SUBSTRING('Hello, World!' FROM -1 FOR 4);
```

```
substring
-----
!
(1 row)
```

Example 3: If **n** is **-1**, an empty string is returned.

```
SELECT SUBSTRING('Hello, World!' FROM -1 FOR -1);
```

```
substring
-----
 
(1 row)
```

Using the BTRIM function

- In the Teradata and Oracle compatibility types, **btrim(string[,characters])** removes the longest string that contains only the characters (a space by default) in the *characters* from a specified position of the *string*.
- In the MySQL compatibility type, it removes the string that is equivalent to characters (a space by default) from a specified position of the *string*.

Using the BTRIM function in the Teradata and Oracle Compatibility Types

```
SELECT BTRIM('xxHello Worldxx', 'xxz');
```

```
btrim
-----
Hello World
(1 row)
```

Using the BTRIM function in the MySQL Compatibility Type

```
SELECT BTRIM('xxHello Worldxx', 'xxz');
```

```
btrim
-----
xxHello Worldxx
(1 row)
```

Using the Log Function

- In the Teradata and Oracle compatibility types, **log(x)** calculates the base-10 logarithm. For instance, **log(100)** equals **log(10,100)**.
- In the MySQL compatibility type, it calculates the natural logarithm using base **e** ($e \approx 2.71828$). For example, **log(100)** roughly matches **log(2.71828,100)**.

Using the Log Function in the Teradata and Oracle Compatibility Types

```
SELECT log(100);
```

```
log
----
2
(1 row)
```

Using the Log Function in the MySQL Compatibility Type

```
SELECT log(100);
```

```
log
-----
4.60517018598809
(1 row)
```

Using the mod(x,0) Function

In the Teradata and Oracle compatibility types, **x** is returned if the divisor of **mod(x)** is **0**. However, in the MySQL compatibility type, this results in an error.

Using the mod(x,0) Function in the Teradata and Oracle Compatibility Types

```
SELECT mod(3,0);
```

```
mod
----
3
(1 row)
```

Using the mod(x,0) Function in the MySQL Compatibility Type

```
SELECT mod(3,0);
```

```
ERROR: division by zero
CONTEXT: referenced column: mod
```

Using the TO_CHAR Function

- In the Teradata compatibility type, the input parameter's maximum value must be a timestamp, not a date. The function returns a date formatted as **YYYY/MM/DD** when GUC parameter [behavior_compat_options](#) is set to **convert_empty_str_to_null_td**.
- In the Oracle compatibility type, the input parameter's maximum value must also be a timestamp, not a date. The returned value is a timestamp.
- In the MySQL compatibility type, the maximum value of the input parameter can be a timestamp or date. The return type is date.

Using the TO_CHAR Function in the Teradata Compatibility Type

```
SET behavior_compat_options = convert_empty_str_to_null_td;
SELECT TO_CHAR(DATE '294276-12-31');
```

```
to_char
-----
294276/12/31
(1 row)
```

```
SELECT TO_CHAR(DATE '5874897-12-31');
```

```
ERROR: date out of range for timestamp
CONTEXT: referenced column: to_char
```

Using the TO_CHAR Function in the Oracle Compatibility Type

```
SELECT TO_CHAR(DATE '294276-12-31');
```

```
to_char
-----
294276-12-31 00:00:00
(1 row)
```

```
SELECT TO_CHAR(DATE '5874897-12-31');
```

```
ERROR: timestamp out of range: "5874897-12-31"
LINE 1: SELECT TO_CHAR(DATE '5874897-12-31');
                        ^
CONTEXT: referenced column: to_char
```

Using the TO_CHAR Function in the MySQL Compatibility Type

```
SELECT TO_CHAR(DATE '294276-12-31');
```

```
to_char
-----
294276-12-31
(1 row)
```

```
SELECT TO_CHAR(DATE '5874897-12-31');
```

```
to_char
-----
5874897-12-31
(1 row)
```

to_date, to_timestamp, and to_number Processing of Empty Strings

- In the Teradata compatibility type, if GUC parameter [behavior_compat_options](#) is set to `convert_empty_str_to_null_td`, these functions return `null`.
- In the Oracle compatibility type, these functions also return `null`.
- In the MySQL compatibility type, `to_date` and `to_timestamp` returns `null`. If the parameter passed to `to_number` is an empty string, `0` is returned.

Empty String Processing in the Teradata and Oracle Compatibility Types (to_date, to_timestamp, and to_number)

```
SET behavior_compat_options = 'convert_empty_str_to_null_td'; --This is only valid for the Teradata-compatible databases.
SELECT TO_DATE('');
SELECT TO_TIMESTAMP('');
SELECT TO_NUMBER('');
```

```
to_number
-----
(1 row)
```

Empty String Processing in the MySQL Compatibility Type (to_date, to_timestamp, and to_number)

```
SELECT TO_DATE('');  
SELECT TO_TIMESTAMP('');  
SELECT TO_NUMBER('');
```

```
to_date  
-----  
  
(1 row)
```

```
to_timestamp  
-----  
  
(1 row)
```

```
to_number  
-----  
0  
  
(1 row)
```

Using the LAST_DAY and NEXT_DAY Functions

- In the Teradata and Oracle compatibility types, these functions return timestamps.
- In the MySQL compatibility type, these functions return dates.

Using the LAST_DAY and NEXT_DAY Functions in the Teradata and Oracle Compatibility Types

```
SELECT last_day(to_date('2024-02-07', 'YYYY-MM-DD')) AS cal_result;
```

```
cal_result  
-----  
2024-02-29 00:00:00  
(1 row)
```

```
SELECT next_day(TIMESTAMP '2024-02-07 00:00:00','Sunday')AS cal_result;
```

```
cal_result  
-----  
2024-02-11 00:00:00  
(1 row)
```

Using the LAST_DAY and NEXT_DAY Functions in the MySQL Compatibility Type

```
SELECT last_day(to_date('2024-02-07', 'YYYY-MM-DD')) AS cal_result;
```



```
cal_result
-----
2024-02-29
(1 row)
```

```
SELECT next_day(TIMESTAMP '2024-02-07 00:00:00','Sunday')AS cal_result;
```

```
cal_result
-----
2024-02-11
(1 row)
```

Using the ADD_MONTHS Function

In the Teradata and Oracle compatibility types, the ADD_MONTHS function returns the timestamp plus integer months.

The ADD_MONTHS function works differently based on the input parameter in the MySQL compatibility type:

- If the input parameter is a date, the function returns a date.
- If the input parameter is a timestamp, the function returns a timestamp.
- If the input parameter is a timestamptz value, the function returns a timestamptz value.

Using the ADD_MONTHS Function in the Teradata and Oracle Compatibility Types

```
SELECT add_months('2024-02-07'::date,3);
```

```
add_months
-----
2024-05-07 00:00:00
(1 row)
```

Using the ADD_MONTHS Function in the MySQL Compatibility Type

```
SELECT add_months('2024-02-07'::date,3);
```

```
add_months
-----
2024-05-07
(1 row)
```

```
SELECT add_months('2024-02-07 00:00:00',3);
```

```
add_months
-----
2024-05-07 00:00:00+08
(1 row)
```

Operator ^

- In the Teradata and Oracle compatibility types, it indicates the exponentiation operation.
- In the MySQL compatibility type, it indicates XOR.

Operator ^ in the Teradata and Oracle Compatibility Types

```
SELECT 10^3;
```

```
?column?
-----
      1000
(1 row)
```

Operator ^ in the MySQL Compatibility Type

```
SELECT 10^3;
```

```
?column?
-----
        9
(1 row)
```

GREATEST and LEAST Expressions

- In the Teradata and Oracle compatibility types, **GREATEST** and **LEAST** return the comparison results of all non-null input parameters.
- In the MySQL compatibility type, it returns **null** if an input parameter contains **null**.

Using GREATEST and LEAST in the Teradata and Oracle Compatibility Types

```
SELECT greatest(1,2,3),least(1,2,3),greatest(1,null,3),least(1,null,3);
```

```
greatest | least | greatest | least
-----+-----+-----+-----
          3 |      1 |          3 |      1
(1 row)
```

Using GREATEST and LEAST in the MySQL Compatibility Type

```
SELECT greatest(1,2,3),least(1,2,3),greatest(1,null,3),least(1,null,3);
```

```
greatest | least | greatest | least
-----+-----+-----+-----
          3 |      1 |          |      |
(1 row)
```

CASE and COALESCE Expressions

- The Teradata compatibility type is compatible with Teradata's behavior and allows converting numbers to strings or vice versa. When using **COALESCE** with INT and VARCHAR inputs, both values become VARCHAR.
- In the Oracle compatibility type, an error is reported if these expressions are used.
- The MySQL compatibility type is compatible with MySQL's behavior and allows converting data types like strings into others. When using **COALESCE** with DATE, INT, or VARCHAR inputs, they all convert to VARCHAR.

Using CASE and COALESCE Expressions in the Teradata Compatibility Type

Type resolution for **CASE** and **COALESCE** in the Teradata Compatibility Type

- If all inputs are of the same type, and it is not **unknown**, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text**.
- If inputs are of string type (including **unknown** which is resolved as type **text**) and digit type, resolve as the string type. If the inputs are not of the two types, fail.
- If the non-unknown inputs are all of the same type category, choose the input type which is a preferred type in that category, if there is one.
- Convert all inputs to the selected type. Fail if there is not an implicit conversion from a given input to the selected type.

Example 1: Use type resolution with underspecified types in a union as the first example. Here, the unknown-type literal **'b'** will be resolved to type **text**.

```
SELECT text 'a' AS "text" UNION SELECT 'b';
```

```
text
-----
a
b
(2 rows)
```

Example 2: Use type resolution in a simple union as the second example. The literal **1.2** is of type **numeric**, and the **integer** value **1** can be cast implicitly to **numeric**, so that type is used.

```
SELECT 1.2 AS "numeric" UNION SELECT 1;
```

```
numeric
-----
1
1.2
(2 rows)
```

Example 3: Use type resolution in a transposed union as the third example. Here, since type **real** cannot be implicitly cast to **integer**, but **integer** can be implicitly cast to **real**, the union result type is resolved as **real**.

```
SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
```

```
real
-----
      1
      2.2
(2 rows)
```

Example 4: In the Teradata compatibility type, if input parameters for **COALESCE** are of **int** and **varchar** types, resolve as type **varchar**. In the Oracle compatibility type, an error is reported. Show the execution plan of a statement for querying the types **int** and **varchar** of input parameters for **COALESCE**.

```
EXPLAIN VERBOSE select coalesce(a, b) FROM td_table;
```

```
QUERY PLAN
-----
id | operation | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+-----
1 | -> Data Node Scan on "__REMOTE_FQS_QUERY__" | 0 | | 0 | 0.00

Targetlist Information (identified by plan id)
-----
1 --Data Node Scan on "__REMOTE_FQS_QUERY__"
   Output: (COALESCE((td_table.a)::character varying, td_table.b))
   Node/s: All datanodes (v3_logical, bucket:24)
   Remote query: SELECT COALESCE(a::character varying, b) AS "coalesce" FROM public.td_table
```

Using CASE and COALESCE Expressions in the Oracle Compatibility Type

In the Teradata compatibility type, if input parameters for **COALESCE** are of **int** and **varchar** types, resolve as type **varchar**. In the Oracle compatibility type, an error is reported. Show the execution plan of a statement for querying the types **int** and **varchar** of input parameters for **COALESCE**.

```
EXPLAIN VERBOSE select coalesce(a, b) FROM ora_table;
```

```
ERROR: COALESCE types integer and character varying cannot be matched
LINE 1: EXPLAIN VERBOSE select coalesce(a, b) FROM ora_table;
                                   ^
CONTEXT: referenced column: coalesce
```

Using CASE and COALESCE Expressions in the MySQL Compatibility Type

The MySQL compatibility type is compatible with behavior of MySQL and allows converting data between string and other types. When using **COALESCE** with inputs of DATE, INT, or VARCHAR types, all values are treated as VARCHAR.

6 DWS Database Security Management

6.1 DWS User and Permissions Management

6.1.1 DWS Database User Types

Without separation of permissions, DWS supports two types of database accounts: administrator and common user. For details about user types and permissions under separation of permissions, see [Separation of Duties in DWS](#).

- The administrator can manage all common users and databases.
- Common users can connect to and access the database, and perform specific database operations and execute SQL statements after being authorized.

Users are authenticated when they log in to the DWS database. A user can own databases and database objects (such as tables), and grant permissions of these objects to other users and roles. In addition to system administrators, users with the **CREATEDB** attribute can create databases and grant permissions to these databases.

Database User Types

Table 6-1 Database user types

User Type	Description	Allowed Operations	How to Create
Administrator dbadmin	An administrator, also called a system administrator, is an account with the SYSADMIN attribute.	If separation of permissions is not enabled, this account has the highest permission in the system and can perform all operations. The system administrator has the same permissions as the object owner.	- User dbadmin created during cluster creation on the DWS console is a system administrator. - Use the CREATE USER or ALTER USER syntax to create an administrator. <pre>CREATE USER sysadmin WITH SYSADMIN password '{Password}'; ALTER USER u1 SYSADMIN;</pre>
Common user	Common user	- Use a tool to connect to the database. - Have the attributes of specific database system operations, such as CREATEDB, CREATEROLE, and SYSADMIN. - Access database objects. - Run SQL statements.	Run the CREATE USER syntax to create a common user. <pre>CREATE USER u1 PASSWORD '{Password}';</pre>
	Private user	A user created with the INDEPENDENT attribute in non-separation-of-permissions mode. Database administrators can manage (DROP , ALTER , and TRUNCATE) objects of private users but cannot access (INSERT , DELETE , SELECT , UPDATE , COPY , GRANT , REVOKE , and ALTER OWNER) the objects before being authorized.	Use the CREATE USER syntax to create a private user. <pre>CREATE USER user_independent WITH INDEPENDENT IDENTIFIED BY '{Password}';</pre>

6.1.2 DWS Database User Management

You can use **CREATE USER** and **ALTER USER** to create and manage database users.

- In the non-[separation-of-permission](#) mode, a DWS user account can be created and deleted only by a system administrator or a security administrator with the **CREATEROLE** attribute.
- In separation-of-permission mode, a user account can be created only by a security administrator.

Creating a User

The **CREATE USER** statement is used to create a DWS user. After creating a user, you can use the user to connect to the database.

- Create common user **u1** and assign the **CREATEDB** attribute to the user.

```
CREATE USER u1 WITH CREATEDB PASSWORD '{Password}';
```
- To create the system administrator **mydbadmin**, you need to specify the **SYSADMIN** parameter.

```
CREATE USER mydbadmin sysadmin PASSWORD '{Password}';
```
- View the created user in the **PG_USER** view.

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

```
SELECT * FROM pg_authid;
```

Altering User Attributes

The **ALTER USER** statement is used to alter user attributes, such as changing user passwords or permissions.

Example:

- Rename user **u1** to **u2**.

```
ALTER USER u1 RENAME TO u2;
```
- Grant the **CREATEROLE** permission to user **u1**:

```
ALTER USER u1 CREATEROLE;
```
- For details about how to change the user password, see [Setting and Changing a Password](#).

Locking a User

The **ACCOUNT LOCK | ACCOUNT UNLOCK** parameter in the statement is used to lock or unlock a user. A locked user cannot log in to the system. If an account is stolen or illegally accessed, the administrator can manually lock the account. After the account is secured, the administrator can manually unlock the account.

Example:

- To lock user **u1**, run the following command:

```
ALTER USER u1 ACCOUNT LOCK;
```
- To unlock user **u1**, run the following command:

```
ALTER USER u1 ACCOUNT UNLOCK;
```

Deleting a User

The **DROP USER** statement is used to delete one or more DWS users. An administrator can delete an account that is no longer used. Deleted users cannot be restored.

- If multiple users are deleted at the same time, separate them with commas (,).
- After a user is deleted successfully, all the permissions of the user are also deleted.
- When an account to be deleted is in the active state, it is deleted after the session is disconnected.
- When **CASCADE** is specified in the **DROP USER** statement, objects such as tables that depend on the user will be deleted. That is, the objects whose owner is the user are deleted, and the authorizations of other objects to the user are also deleted.

Example:

- -- Delete user **u1**.
`DROP USER u1;`
- Delete account **u2** in a cascading manner.
`DROP USER u2 CASCADE;`

6.1.3 Creating a Custom Password Policy for DWS

When creating or modifying a user, you need to specify a password. DWS has default password complexity requirements. You can also define database account password policies.

Default DWS Password Policy

By default, DWS verifies the password complexity. The default password policy requires that the password:

- Contain 8 to 32 characters.
- Contain at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters.
- Cannot be the same as the user name or the user name in reverse order, case insensitive.
- Cannot be the current password or the current password in reverse order.

User-defined Password Policy

The password policy includes the password complexity requirements, password validity period, password reuse settings, password encryption mode, and password retry and lock policies. Different policy items are controlled by the corresponding GUC parameters. For details, see [Security and Authentication \(postgresql.conf\)](#).

Table 6-2 User-defined password policies and corresponding GUC parameters

Password Policy	Parameter	Description	Value Range	Default Value in DWS
Password complexity check	password_policy	Specifies whether to check the password complexity when a DWS account is created or modified.	Integer, 0 or 1 0 indicates that no password complexity policy is used. Setting this parameter to 0 leads to security risks. You are advised not to set this parameter to 0. 1 indicates that the default password complexity policy is used.	1
Password complexity requirement	password_min_length	Specifies the minimum password length.	An integer ranging from 6 to 999	8
	password_max_length	Specifies the maximum password length.	An integer ranging from 6 to 999	32
	password_min_uppercase	Minimum number of uppercase letters (A-Z)	An integer ranging from 0 to 999 0 means no requirements. 1-999 indicates the minimum number of uppercase letters in the password.	0
	password_min_lowercase	Minimum number of lowercase letters (a-z)	An integer ranging from 0 to 999 0 means no requirements. 1-999 indicates the minimum number of lower letters in the password.	0

Password Policy	Parameter	Description	Value Range	Default Value in DWS
	password_min_digital	Minimum number of digits (0-9)	An integer ranging from 0 to 999 • 0 means no requirements. • 1-999 indicates the minimum number of digits in the password.	0
	password_min_special	Minimum number of special characters (Table 6-3 lists the special characters.)	An integer ranging from 0 to 999 • 0 means no requirements. • 1-999 indicates the minimum number of special characters in the password.	0
Password validity	password_effect_time	Password validity period When the number of days in advance a user is notified that the password is about to expire reaches the value of password_notify_time , the system prompts the user to change the password when the user logs in to the database.	The value is a floating point number ranging from 0 to 999. The unit is day. • 0 indicates the validity period is disabled. • A floating point number from 1 to 999 indicates the validity period of the password. When the password is about to expire or has expired, the system prompts the user to change the password.	90

Password Policy	Parameter	Description	Value Range	Default Value in DWS
	password_notify_time	Specifies for how many days you are reminded of the password expiry.	The value is an integer ranging from 0 to 999. The unit is day. • 0 indicates the reminder is disabled. • A value ranging from 1 to 999 indicates the number of days prior to password expiration that a user will receive a notification.	7
Password reuse settings	password_reuse_time	Specifies the number of days after which the password cannot be reused.	A Floating point number ranging from 0 to 3650. The unit is day. • 0 indicates that the password reuse days are not checked. • A positive number indicates that the new password cannot be chosen from passwords in history that are newer than the specified number of days.	60
	password_reuse_max	Specifies the number of the most recent passwords that the new password cannot be chosen from.	An integer ranging from 0 to 1000 • 0 indicates that the password reuse times are not checked. • A positive number indicates that the new password cannot be chosen from the specified number of the most recent passwords.	0

Password Policy	Parameter	Description	Value Range	Default Value in DWS
Encryption mode	password_encryption_type	Specifies the password storage encryption mode.	0, 1, 2 • 0 indicates that passwords are encrypted in MD5 mode. The password is encrypted using MD5. This mode is not recommended for users. • 1 indicates that passwords are encrypted with SHA-256, which is compatible with the MD5 user authentication method of the PostgreSQL client. The password is stored in ciphertext encrypted by MD5 and SHA256. • 2 indicates that passwords are encrypted using SHA-256. The password is encrypted using SHA256.	1

Password Policy	Parameter	Description	Value Range	Default Value in DWS
Retry and lock	password_lock_time	Specifies the duration for a locked account to be automatically unlocked.	A Floating point number ranging from 0 to 365. The unit is day. • 0 indicates that the account is not automatically locked if the password verification fails. • A positive number indicates the duration after which a locked account is automatically unlocked. NOTE The integral part of the value of the password_lock_time parameter indicates the number of days and its decimal part can be converted into hours, minutes, and seconds.	1
	failed_login_attempts	If the number of incorrect password attempts reaches the value of failed_login_attempts, the account is locked and will be automatically unlocked in X (which indicates the value of password_lock_time) seconds.	An integer ranging from 0 to 1000 • 0 indicates that the automatic locking function does not take effect. • A positive number indicates that an account is locked when the number of incorrect password attempts reaches the value of failed_login_attempts.	10

Table 6-3 Special characters

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

Example 2: Configure password_effect_time.

1. Run the following command to connect to the database:

```
gsql -d postgres -p 25308
```

postgres indicates the name of the database you want to connect. **25308** indicates the CN port.

If information similar to the following is displayed, the connection succeeds:

```
gsql ((GaussDB x.x.x build 39137c2d) compiled at 2022-04-01 15:43:11 commit 3629 last mr 5138
release)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

postgres=#
```

2. View the configured parameter.

```
SHOW password_effect_time;
password_effect_time
-----
90
(1 row)
```

If the command output is not **90**, run the **\q** command to exit the database.

3. If **90** is not displayed, run the following command to set the parameter to **90** (**0** is not recommended):

```
gs_guc reload -Z coordinator -Z datanode -N all -I all -c "password_effect_time = 90"
```

Setting and Changing a Password

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

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

```
ALTER USER user1 IDENTIFIED BY 'newpassword' REPLACE 'oldpassword';
```

NOTE

The password must meet requirements in [Default DWS Password Policy](#), or the execution will fail.

- An administrator can change its own password and other accounts' passwords. With the permission for changing other accounts' passwords, the administrator can resolve a login failure when a user forgets its password.

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

```
ALTER USER joe IDENTIFIED BY 'password';
```

NOTE

- System administrators are not allowed to change passwords for each other.
- When a system administrator changes the password of a common user, the original password is not required.
- However, when a system administrator changes its own password, the original password is required.

- Password verification

Password verification is required when you set the user or role in the current session. If the entered password is inconsistent with the stored password of the user, an error is reported.

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

```
SET ROLE joe PASSWORD 'password';
```

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

```
SET ROLE
```

6.1.4 DWS Database Permissions Management

Permission Overview

Permissions are used to control whether a user is allowed to access a database object (including schemas, tables, functions, and sequences) to perform operations such as adding, deleting, modifying, querying, and creating a database object.

Permissions management in DWS falls into three categories:

- System permissions

System permissions are also called user attributes, including **SYSADMIN**, **CREATEDB**, **CREATEROLE**, **AUDITADMIN**, and **LOGIN**.

They can be specified only by the **CREATE ROLE** or **ALTER ROLE** syntax. The **SYSADMIN** permission can be granted and revoked using **GRANT ALL PRIVILEGE** and **REVOKE ALL PRIVILEGE**, respectively. System permissions cannot be inherited by a user from a role, and cannot be granted using **PUBLIC**.

- Object permissions

Permissions on a database object (table, view, column, database, function, schema, or tablespace) can be granted to a role or user. The **GRANT** command can be used to grant permissions to a user or role. These permissions granted are added to the existing ones.

- Permissions

Grant a role's or user's permissions to one or more roles or users. In this case, every role or user can be regarded as a set of one or more database permissions.

If **WITH ADMIN OPTION** is specified, the member can in turn grant permissions in the role to others, and revoke permissions in the role as well. If

a role or user granted with certain permissions is changed or revoked, the permissions inherited from the role or user also change.

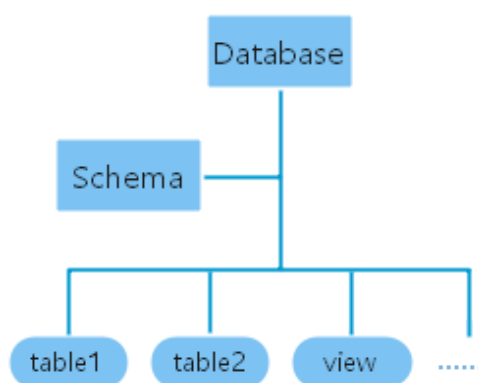
A database administrator can grant permissions to and revoke them from any role or user. Roles having **CREATEROLE** permission can grant or revoke membership in any role that is not an administrator.

Hierarchical Permission Management

DWS implements a hierarchical permission management on databases, schemas, and data objects.

- Databases cannot communicate with each other and share very few resources. Their connections and permissions can be isolated. The database cluster has one or more named databases. Users and roles are shared within the entire cluster, but their data is not shared. That is, a user can connect to any database, but after the connection is successful, any user can access only the database declared in the connection request.
- Schemas share more resources than databases do. User permissions on schemas and subordinate objects can be flexibly configured using the **GRANT** and **REVOKE** syntax. Each database has one or more schemas. Each schema contains various types of objects, such as tables, views, and functions. To access an object contained in a specified schema, a user must have the **USAGE** permission on the schema.
- After an object is created, by default, only the object owner or system administrator can query, modify, and delete the object. To access a specific database object, for example, **table1**, other users must be granted the **CONNECT** permission of database, the **USAGE** permission of schema, and the **SELECT** permission of **table1**. To access an object at the bottom layer, a user must be granted the permission on the object at the upper layer. To create or delete a schema, you must have the **CREATE** permission on its database.

Figure 6-1 Hierarchical Permission Management



Roles

The permission management model of DWS is a typical implementation of the role-based access control (RBAC). It manages users, roles, and permissions through this model.

A role is a set of permissions.

- The concept of "user" is equivalent to that of "role". The only difference is that "user" has the **login** permission while "role" has the **nologin** permission.
- Roles are assigned with different permissions based on their responsibilities in the database system. A role is a set of database permissions and represents the behavior constraints of a database user or a group of data users.
- Roles and users can be converted. You can use **ALTER** to assign the **login** permission to a role.
- After a role is granted to a user through **GRANT**, the user will have all the permissions of the role. It is recommended that roles be used to efficiently grant permissions. For example, you can create different roles of design, development, and maintenance personnel, grant the roles to users, and then grant specific data permissions required by different users. When permissions are granted or revoked at the role level, these permission changes take effect for all the members of the role.
- In non-separation-of-duty scenarios, a role can be created, modified, and deleted only by a system administrator or a user with the **CREATEROLE** attribute. In separation-of-duty scenarios, a role can be created, modified, and deleted only by a user with the **CREATEROLE** attribute.

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

```
SELECT * FROM PG_ROLES;
```

For details about how to create, modify, and delete roles, see "CREATE ROLE", "ALTER ROLE", and "DROP ROLE" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

Preset Roles

DWS provides a group of preset roles. Their names start with **gs_role_**. These roles allow access to operations that require high permissions. You can grant these roles to other users or roles in the database for them to access or use specific information and functions. Exercise caution and ensure security when using preset roles.

The following table describes the permissions of preset roles.

Table 6-4 Permissions of preset roles

Role	Permission
gs_role_signal_backend	Invokes functions such as pg_cancel_backend , pg_terminate_backend , pg_terminate_query , pg_cancel_query , pgxc_terminate_query , and pgxc_cancel_query to cancel or terminate sessions, excluding those of the initial users.

Role	Permission
gs_role_read_all_stats	Reads the system status view and uses various extension-related statistics, including information that is usually visible only to system administrators. For example: Resource management views: - pgxc_wlm_operator_history - pgxc_wlm_operator_info - pgxc_wlm_operator_statistics - pgxc_wlm_session_info - pgxc_wlm_session_statistics - pgxc_wlm_workload_records - pgxc_workload_sql_count - pgxc_workload_sql_elapse_time - pgxc_workload_transaction Status information views: - pgxc_stat_activity - pgxc_get_table_skewness - table_distribution - pgxc_total_memory_detail - pgxc_os_run_info - pg_nodes_memory - pgxc_instance_time - pgxc_redo_stat
gs_role_analyze_any	A user with the system-level ANALYZE permission can skip the schema permission check and perform ANALYZE on all tables.
gs_role_vacuum_any	A user with the system-level VACUUM permission can skip the schema permission check and perform ANALYZE on all tables.
gs_redaction_policy	A user with the permission to create, modify, and delete data masking policies and can execute CREATE ALTER DROP REDACTION POLICY on all tables. Clusters of 9.1.0 and later versions support this function.

Restrictions on using preset roles:

- **gs_role_** is the name field dedicated to preset roles in the database. Do not create users or roles starting with **gs_role_** or rename existing users or roles starting with **gs_role_**.
- Do not perform **ALTER** or **DROP** operations on preset roles.
- By default, a preset role does not have the **LOGIN** permission, so there is no preset login password for the role.

- The gsql meta-commands `\du` and `\dg` do not display information about preset roles. However, if **PATTERN** is specified, information about preset roles will be displayed.
- If the separation of permissions is disabled, the system administrator and users with the **ADMIN OPTION** permission of preset roles are allowed to perform GRANT and REVOKE operations on preset roles. If the separation of permissions is enabled, the security administrator (with the **CREATEROLE** attribute) and users with the **ADMIN OPTION** permission of preset roles are allowed to perform GRANT and REVOKE operations on preset roles. Example:
GRANT gs_role_signal_backend TO user1;
REVOKE gs_role_signal_backend FROM user1;

Granting or Revoking Permissions

A user who creates an object is the owner of this object. By default, [Separation of Duties in DWS](#) is disabled after cluster installation. A database system administrator has the same permissions as object owners.

After an object is created, only the object owner or system administrator can query, modify, and delete the object, and grant permissions for the object to other users through **GRANT** by default. To enable a user to use an object, the object owner or administrator can run the **GRANT** or **REVOKE** command to grant permissions to or revoke permissions from the user or role.

- Run the **GRANT** statement to grant permissions.
For example, grant the permission of schema **myschema** to role **u1**, and grant the **SELECT** permission of table **myschema.t1** to role **u1**.
GRANT USAGE ON SCHEMA *myschema* TO *u1*;
GRANT *SELECT* ON TABLE *myschema.t1* TO *u1*;
- Run the **REVOKE** command to revoke a permission that has been granted.
For example, revoke all permissions of user **u1** on the **myschema.t1** table.
REVOKE ALL PRIVILEGES ON *myschema.t1* FROM *u1*;

6.1.5 Separation of Duties in DWS

By default, the system administrator with the **SYSADMIN** attribute has the highest permission in the system. To avoid risks caused by centralized permissions, you can enable the separation of permissions to delegate system administrator permissions to security administrators and audit administrators.

- After the separation of permissions is enabled, a system administrator does not have the **CREATEROLE** attribute (security administrator) and **AUDITADMIN** attribute (audit administrator). That is, you do not have the permissions for creating roles and users and the permissions for viewing and maintaining database audit logs. For more information about the **CREATEROLE** and **AUDITADMIN** attributes, see "CREATE ROLE" in the *Data Warehouse Service (DWS) SQL Syntax Reference*.
- After the separation of permissions is enabled, system administrators have the permissions only for the objects owned by them.

For how to configure permission separation, see "Configuring Separation of Duties for the DWS Cluster" in the *Data Warehouse Service (DWS) User Guide*.

For details about permission changes before and after enabling the separation of permissions, see [Table 6-5](#) and [Table 6-6](#).

Table 6-5 Default user permissions

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespace	Can create, modify, delete, access, and allocate tablespaces.	Cannot create, modify, delete, or allocate tablespaces, with authorization required for accessing tablespaces.		
Table	Has permissions for all tables.	Has permissions for its own tables, but does not have permissions for other users' tables.		
Index	Can create indexes on all tables.	Can create indexes on their own tables.		
Schema	Has permissions for all schemas.	Has all permissions for its own schemas, but does not have permissions for other users' schemas.		
Function	Has permissions for all functions.	Has permissions for its own functions, has the call permission for other users' functions in the public schema, but does not have permissions for other users' functions in other schemas.		
Customized view	Has permissions for all views.	Has permissions for its own views, but does not have permissions for other users' views.		
System catalog and system view	Has permissions for querying all system catalogs and views.	Has permissions for querying only some system catalogs and views. For details, see DWS System Catalogs and Views .		

Table 6-6 Changes in permissions after the separation of permissions

Object	System Administrator	Security Administrator	Audit Administrator	Common User
Tablespace	No change	No change		
Table	Permissions reduced Has all permissions for its own tables, but does not have permissions for other users' tables in their schemas.	No change		

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

6.2 DWS Sensitive Data Management

6.2.1 DWS Row-Level Access Control

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

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

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

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

```
-- Create users alice, bob, and peter.
CREATE ROLE alice PASSWORD 'password';
CREATE ROLE bob PASSWORD 'password';
CREATE ROLE peter PASSWORD 'password';

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

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

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

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

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

-- View table details.
\d+ all_data
```

Column	Type	Modifiers	Storage	Stats target	Description
id	integer		plain		
role	character varying(100)		extended		
data	character varying(100)		extended		

```
Row Level Security Policies:
  POLICY "all_data_rls"
    USING (((role)::name = "current_user"()))
Has OIDs: no
Distribute By: HASH(id)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no, enable_rowsecurity=true

-- Switch to user alice and run SELECT * FROM all_data.
SET ROLE alice PASSWORD 'password';
SELECT * FROM all_data;
id | role | data
---+-----+-----
 1 | alice | alice data
(1 row)

EXPLAIN(COSTS OFF) SELECT * FROM all_data;
QUERY PLAN

-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on all_data
    Filter: ((role)::name = 'alice'::name)
Notice: This query is influenced by row level security feature
(5 rows)

-- Switch to user peter and run SELECT * FROM .all_data.
SET ROLE peter PASSWORD 'password';
SELECT * FROM all_data;
```

```
id | role | data
-----+-----+-----
 3 | peter | peter data
(1 row)

EXPLAIN(COSTS OFF) SELECT * FROM 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)
```

6.2.2 DWS Data Masking

DWS provides the column-level dynamic data masking (DDM) function. For sensitive data (such as the ID card number, mobile number, and bank card number), the DDM function is used to redact the original data to protect data security and user privacy.

- Creating a data masking policy for a table
DWS uses the **CREATE REDACTION POLICY** syntax to create a data masking policy on a table (Do not perform masking), **MASK_FULL** (Mask data into a fixed value), and **MASK_PARTIAL** (Perform partial masking based on the character type, numeric type, or time type.) to specify the application scope of the masking policy.
- Modifying the data masking policy of a table
The **ALTER REDACTION POLICY** syntax is used to modify the expression for enabling a masking policy, rename a masking policy, and add, modify, or delete masked columns.
- Deleting the masking policy of a table
The **DROP REDACTION POLICY** syntax is used to delete the masking function information of a masking policy on all columns of a table.
- Viewing the masking policy and masked columns
Masking policy information is stored in the system catalog [PG_REDACTION_POLICY](#), and masked column information is stored in the system catalog [PG_REDACTION_COLUMN](#). You can view information about the masking policy and masked columns in the system views [REDACTION_POLICIES](#) and [REDACTION_COLUMNS](#).

Precautions

- The data masking feature is controlled by **feature_support_options**. If error information "REDACTION POLICY is not yet supported, please add enable_data_redaction into feature_support_options." or "Cannot use the feature of data redaction, please drop existed redaction policy or add enable_data_redaction into feature_support_options." is displayed, the data masking feature is not enabled. In this case, you need to contact technical support to set **feature_support_options**.
- Generally, you can run the SELECT statement to view the data masking result. If a statement has the following features, sensitive data may be deliberately obtained. In this case, an error will be reported during statement execution.

- The GROUP BY clause references the Target Entry containing masked columns as the target column.
- DISTINCT works on the output masked columns.
- The statement contains CTE.
- Operations on sets are involved.
- The target columns of a subquery are not masked columns of the base table, but the expressions or function calls for masked columns of the base table.
- You can use COPY TO or GDS to export the masked data. Due to the irreversibility of the data masking, secondary masking of the data is meaningless.
- Do not set target columns of UPDATE, MERGE INTO, and DELETE statements to masked columns.
- The UPSERT statement allows you to insert update data through EXCLUDED. If data in the base table is updated by referencing masked columns, the data may be modified by mistake. As a result, an error will be reported during the execution.
- In the 8.2.1 cluster version, multiple masking policies can be created for the same table to implement diversified sensitive data classification. The principles for selecting and applying masking policies are as follows:
 - Select the policy with the largest **policy_order** among multiple candidate policies that meet the requirements of the current session. A larger **policy_order** indicates a later creation.
 - During data masking, the DML statement inherits only the policy with the largest **policy_order**.

Examples

The following uses the employee table **emp**, table owner **alice**, and roles **matu** and **july** as an example to illustrate the data masking process. The **emp** table contains private data such as the employee name, mobile number, email address, bank card number, and salary.

Step 1 After connecting to the database as the administrator, create roles **alice**, **matu**, and **july**.

```
CREATE ROLE alice PASSWORD 'password';
CREATE ROLE matu PASSWORD 'password';
CREATE ROLE july PASSWORD 'password';
```

Step 2 Grant schema permissions on the current database to **alice**, **matu**, and **july**.

```
GRANT ALL PRIVILEGES on schema public to alice,matu,july;
```

Step 3 Switch to role **alice**, create the **emp** table, and insert three pieces of employee information.

```
SET ROLE alice PASSWORD 'password';

CREATE TABLE emp(id int, name varchar(20), phone_no varchar(11), card_no number, card_string
varchar(19), email text, salary numeric(100, 4), birthday date);

INSERT INTO emp VALUES(1, 'anny', '13420002340', 1234123412341234, '1234-1234-1234-1234',
'smithWu@163.com', 10000.00, '1999-10-02');
INSERT INTO emp VALUES(2, 'bob', '18299023211', 3456345634563456, '3456-3456-3456-3456',
'66allen_mm@qq.com', 9999.99, '1989-12-12');
```



```
INSERT INTO emp VALUES(3, 'cici', '15512231233', NULL, NULL, 'jonesishere@sina.com', NULL, '1992-11-06');
```

Step 4 **alice** grants the read permission on the **emp** table to **matu** and **july**.

```
GRANT SELECT ON emp TO matu, july;
```

Step 5 Create the masking policy **mask_emp**: Only user **alice** can view all employee information. User **matu** and **july** cannot view employee bank card numbers and salary data. The **card_no** column is of the numeric type and all of its data is masked into 0 by the **MASK_FULL** function. The **card_string** column is of the character type and part of its data is masked by the **MASK_PARTIAL** function based on the specified input and output formats. The **salary** column is of the numeric type and the **MASK_PARTIAL** function is used to mask all digits before the penultimate digit using the number 9.

```
CREATE REDACTION POLICY mask_emp ON emp WHEN (current_user IN ('matu', 'july'))  
ADD COLUMN card_no WITH mask_full(card_no),  
ADD COLUMN card_string WITH mask_partial(card_string, 'VVVVVVVVVVVVVVVVVV', 'VVVV-VVVV-VVVV-VVVV', '#', 1, 12),  
ADD COLUMN salary WITH mask_partial(salary, '9', 1, length(salary) - 2);
```

Step 6 Switch to **matu** and **july** and view the employee table **emp**.

```
SET ROLE matu PASSWORD 'password';  
SELECT * FROM emp;  
id | name | phone_no | card_no | card_string | email | salary | birthday  
-----  
1 | anny | 13420002340 | 0 | #####-#####-1234 | smithWu@163.com | 99999.9990 | 1999-10-02 00:00:00  
2 | bob | 18299023211 | 0 | #####-#####-3456 | 66allen_mm@qq.com | 9999.9990 | 1989-12-12 00:00:00  
3 | cici | 15512231233 | | | jonesishere@sina.com | | 1992-11-06 00:00:00  
(3 rows)  
  
SET ROLE july PASSWORD 'password';  
SELECT * FROM emp;  
id | name | phone_no | card_no | card_string | email | salary | birthday  
-----  
1 | anny | 13420002340 | 0 | #####-#####-1234 | smithWu@163.com | 99999.9990 | 1999-10-02 00:00:00  
2 | bob | 18299023211 | 0 | #####-#####-3456 | 66allen_mm@qq.com | 9999.9990 | 1989-12-12 00:00:00  
3 | cici | 15512231233 | | | jonesishere@sina.com | | 1992-11-06 00:00:00  
(3 rows)
```

Step 7 If you want **matu** to have the permission to view all employee information, but do not want **july** to have. In this case, you only need to modify the effective scope of the policy.

```
SET ROLE alice PASSWORD 'password';  
ALTER REDACTION POLICY mask_emp ON emp WHEN(current_user = 'july');
```

Step 8 Switch to users **matu** and **july** and view the **emp** table again, respectively.

```
SET ROLE matu PASSWORD 'password';  
SELECT * FROM emp;  
id | name | phone_no | card_no | card_string | email | salary | birthday  
-----  
1 | anny | 13420002340 | 1234123412341234 | 1234-1234-1234-1234 | smithWu@163.com | 10000.0000 | 1999-10-02 00:00:00  
2 | bob | 18299023211 | 3456345634563456 | 3456-3456-3456-3456 | 66allen_mm@qq.com | 9999.9900 | 1989-12-12 00:00:00  
3 | cici | 15512231233 | | | jonesishere@sina.com | | 1992-11-06 00:00:00  
(3 rows)  
  
SET ROLE july PASSWORD 'password';  
SELECT * FROM emp;  
id | name | phone_no | card_no | card_string | email | salary | birthday
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
1 | anny | 13420002340 | 0 | #####-#####-1234 | smithWu@163.com | 99999.9990 |
1999-10-02 00:00:00
2 | bob | 18299023211 | 0 | #####-#####-3456 | 66allen_mm@qq.com | 9999.9990 |
1989-12-12 00:00:00
3 | cici | 15512231233 | | | jonesishere@sina.com | | 1992-11-06 00:00:00
(3 rows)

```

Step 9 The information in the **phone_no**, **email**, and **birthday** columns is private data. Update masking policy **mask_emp** and add three masked columns.

```

SET ROLE alice PASSWORD 'password';
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN phone_no WITH mask_partial(phone_no, '*',
4);
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN email WITH mask_partial(email, '*', 1,
position('@' in email));
ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN birthday WITH mask_full(birthday);

```

Step 10 Switch to **july** and view data in the **emp** table.

```

SET ROLE july PASSWORD 'password';
SELECT * FROM emp;
id | name | phone_no | card_no | card_string | email | salary | birthday
+-----+-----+-----+-----+-----+-----+-----+-----+
1 | anny | 134***** | 0 | #####-#####-1234 | *****163.com | 99999.9990 | 1970-01-01
00:00:00
2 | bob | 182***** | 0 | #####-#####-3456 | *****qq.com | 9999.9990 | 1970-01-01
00:00:00
3 | cici | 155***** | | | *****sina.com | | 1970-01-01 00:00:00
(3 rows)

```

Step 11 Query **redaction_policies** and **redaction_columns** to view details about the current redaction policy **mask_emp**.

```

SELECT * FROM redaction_policies;
object_schema | object_owner | object_name | policy_name | expression | enable |
policy_description | inherited
+-----+-----+-----+-----+-----+-----+-----+
public | alice | emp | mask_emp | ("current_user"() = 'july'::name) | t |
f
(1 row)

SELECT object_name, column_name, function_info FROM redaction_columns;
object_name | column_name | function_info
+-----+-----+-----+
emp | card_no | mask_full(card_no)
emp | card_string | mask_partial(card_string, 'VVVVVVVVVVVVVVVVVVVVV'::text, 'VVVV-VVVV-VVVV-
VVVV'::text, '#'::text, 1, 12)
emp | email | mask_partial(email, '*'::text, 1, "position"(email, '@'::text))
emp | salary | mask_partial(salary, '9'::text, 1, (length((salary)::text) - 2))
emp | birthday | mask_full(birthday)
emp | phone_no | mask_partial(phone_no, '*'::text, 4)
(6 rows)

```

Step 12 Add the **salary_info** column. To replace the salary information in text format with ***,***, you can create a user-defined masking function. In this step, you can use the PL/pgSQL to define the masking function **mask_regexp_salary**. To create a masking column, you simply need to customize the function name and parameter list. For details, see [DWS User-Defined Functions](#).

```

SET ROLE alice PASSWORD 'password';

ALTER TABLE emp ADD COLUMN salary_info TEXT;
UPDATE emp SET salary_info = salary::text;

CREATE FUNCTION mask_regexp_salary(salary_info text) RETURNS text AS
$$
SELECT regexp_replace($1, '[0-9]+'::text, '*'::text, 'g');

```

```
$$
LANGUAGE SQL
STRICT SHIPPABLE;

ALTER REDACTION POLICY mask_emp ON emp ADD COLUMN salary_info WITH
mask_regexp_salary(salary_info);

SET ROLE july PASSWORD 'password';
SELECT id, name, salary_info FROM emp;
id | name | salary_info
---+-----+-----
1 | anny | **
2 | bob | **
3 | cici |
(3 rows)
```

Step 13 If there is no need to set a redaction policy for the **emp** table, delete redaction policy **mask_emp**.

```
SET ROLE alice PASSWORD 'password';
DROP REDACTION POLICY mask_emp ON emp;
```

----End

6.2.3 Encrypting and Decrypting DWS Strings

DWS allows you to use the following functions to encrypt and decrypt strings:

- `gs_encrypt(encryptstr, keystr, cryptotype, cryptomode, hashmethod)`
Description: Encrypts an **encryptstr** string using the **keystr** key based on the encryption algorithm specified by **cryptotype** and **cryptomode** and the HMAC algorithm specified by **hashmethod**, and returns the encrypted string. **cryptotype** can be **aes128**, **aes192**, and **aes256**. **cryptomode** is **cbc**. **hashmethod** can be **sha256**, **sha384**, **sha512**, or **sm3**. Currently, the following types of data can be encrypted: numerals supported in the database; character type; RAW in binary type; and DATE, TIMESTAMP, and SMALLDATETIME in date/time type. The **keystr** length is related to the encryption algorithm and contains 1 to **KeyLen** bytes. If **cryptotype** is **aes128**, **KeyLen** is 16; if **cryptotype** is **aes192**, **KeyLen** is 24; if **cryptotype** is **aes256**, **KeyLen** is 32.

Return type: text

Length of the return value: at least $4 \times ((\text{maclen} + 56)/3)$ bytes and no more than $4 \times ((\text{Len} + \text{maclen} + 56)/3)$ bytes, where **Len** indicates the string length (in bytes) before the encryption and **maclen** indicates the length of the HMAC value. If **hashmethod** is **sha256** or **sm3**, **maclen** is 32; if **hashmethod** is **sha384**, **maclen** is 48; if **hashmethod** is **sha512**, **maclen** is 64. That is, if **hashmethod** is **sha256** or **sm3**, the returned string contains 120 to $4 \times ((\text{Len} + 88)/3)$ bytes; if **hashmethod** is **sha384**, the returned string contains 140 to $4 \times ((\text{Len} + 104)/3)$ bytes; if **hashmethod** is **sha512**, the returned string contains 160 to $4 \times ((\text{Len} + 120)/3)$ bytes.

Example:

```
SELECT gs_encrypt('DWS', '1234', 'aes128', 'cbc', 'sha256');
          gs_encrypt
-----
AAAAAAAAAAAAArYr4+WFD1Tew23KhoB7H7cPComVRqy3O/3CnSglER3lAw662L6Qfl
+oJFvHTRnqhS10KYbE+Xui5kYU5Hn56l7QaG6VuCKG84uZ9X8+nYXw==
(1 row)
```

 NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
- Do not use the **ge_encrypt** and **gs_encrypt_aes128** functions for the same data table.
- **gs_decrypt**(decryptstr, keystr, cryptotype, cryptomode, hashmethod)
Description: Decrypts a **decryptstr** string using the **keystr** key based on the encryption algorithm specified by **cryptotype** and **cryptomode** and the HMAC algorithm specified by **hashmethod**, and returns the decrypted string. The **keystr** used for decryption must be consistent with that used for encryption. **keystr** cannot be empty.

Return type: text

Example:

```
SELECT gs_decrypt('AAAAAAAAAArYr4+WFD1Tew23KhoB7H7cPComVRqy3O/3CnSgLER3IAw662L6Qfl
+ojFvHTRnqhS10KYbE+Xui5kYU5Hn56l7QaG6VuCKG84uZ9X8+nYXw=','1234','aes128','cbc',
'sha256');
gs_decrypt
-----
DWS
(1 row)
```

 NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
- This function works with the **gs_encrypt** function, and the two functions must use the same encryption algorithm and HMAC algorithm.
- **gs_encrypt_aes128**(encryptstr,keystr)
Description: Encrypts **encryptstr** strings using **keystr** as the key and returns encrypted strings. The length of **keystr** ranges from 1 to 16 bytes. Currently, the following types of data can be encrypted: numerals supported in the database; character type; RAW in binary type; and DATE, TIMESTAMP, and SMALLDATETIME in date/time type.

Return type: text

Length of the return value: At least 92 bytes and no more than $(4 * [Len / 3] + 68)$ bytes, where *Len* indicates the length of the data before encryption (unit: byte).

Example:

```
SELECT gs_encrypt_aes128('DWS','1234');
gs_encrypt_aes128
-----
ZrCp794vO5I9qJ+jHFf/sQqRyMBy0lKIDGP5S8RJXzgmPXoa/
e4EgmK82P5y5xe1bOXbJeoNxyHagK9OhPVVeJDbn/M=
(1 row)
```

 NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
- Do not use the **ge_encrypt** and **gs_encrypt_aes128** functions for the same data table.
- **gs_decrypt_aes128(decryptstr,keyststr)**
Description: Decrypts a **decryptstr** string using the **keyststr** key and returns the decrypted string. The **keyststr** used for decryption must be consistent with that used for encryption. **keyststr** cannot be empty.

Return type: text

Example:

```
SELECT gs_decrypt_aes128('ZrCp794vO5I9qJ+jHfF/sQqRyMBy0lKIDGP5S8RjXzgrpXoa/
e4EgmK82P5y5xe1bOXbJeoNxyHagK9OhPVVeJDbn/M='; '1234');
gs_decrypt_aes128
```

```
-----
DWS
(1 row)
```

 NOTE

- A decryption password is required during the execution of this function. For security purposes, the gsql tool does not record this function in the execution history. That is, the execution history of this function cannot be found in **gsql** by paging up and down.
- This function works with the **gs_encrypt_aes128** function.
- **gs_hash(hashstr, hashmethod)**

Description: Obtains the digest string of a **hashstr** string based on the algorithm specified by **hashmethod**. **hashmethod** can be **sha256**, **sha384**, **sha512**, or **sm3**.

Return type: text

Length of the return value: 64 bytes if **hashmethod** is **sha256** or **sm3**; 96 bytes if **hashmethod** is **sha384**; 128 bytes if **hashmethod** is **sha512**

Example:

```
SELECT gs_hash('DWS', 'sha256');
gs_hash
-----
1d595d9e342bccc3b158fabfcdfce54d9f821cf9c7461c5d3b6c0157c442f94
(1 row)
```

- **md5(string)**
Description: Encrypts a string in MD5 mode and returns a value in hexadecimal form.

 NOTE

MD5 is insecure and is not recommended.

Return type: text

Example:

```
SELECT md5('ABC');
md5
-----
902fbdd2b1df0c4f70b4a5d23525e932
(1 row)
```

6.2.4 Using pgcrypto to Encrypt DWS Data

DWS clusters of version 8.2.0 or later provide a built-in cryptographic module pgcrypto. The pgcrypto module allows database users to store certain columns of data after encryption, enhancing sensitive data security. Users without the encryption key cannot read the encrypted data stored in DWS.

The pgcrypto function runs inside database servers, which means that all data and passwords are transmitted in plaintext between pgcrypto and client applications. For security purposes, you are advised to use the SSL connection between the client and the DWS server.

The functions in the pgcrypto module are as follows.

General Hash Functions

- digest()

The digest() function can generate binary hash values by using a specified algorithm. The syntax is as follows:

```
digest(data text, type text) returns bytea  
digest(data bytea, type text) returns bytea
```

data indicates the original data, and **type** indicates the encryption algorithm (**md5**, **sha1**, **sha224**, **sha256**, **sha384**, **sha512**, or **sm3**). The return value of the function is a binary string.

Example:

Use the digest() function to encrypt the DWS string using SHA256 for storage.

```
select digest('DWS', 'sha256');  
      digest  
-----  
 \x1d595d9e342bcc3b158fabfcdfece54d9f821cf9c7461c5d3b6c0157c442f94  
(1 row)
```

- hmac()

The hmac() function can calculate the MAC value for data with a key by using a specified algorithm. The syntax is as follows:

```
hmac(data text, key text, type text) returns bytea  
hmac(data bytea, key bytea, type text) returns bytea
```

data indicates the original data, **key** indicates the encryption key, and **type** indicates the encryption algorithm (**md5**, **sha1**, **sha224**, **sha256**, **sha384**, **sha512**, or **sm3**). The return value of the function is a binary string.

Example:

Use **key123** and the SHA256 algorithm to calculate the MAC value for the string **DWS**.

```
select hmac('DWS', 'key123', 'sha256');  
      hmac  
-----  
 \xa2ee3b0d627a93166ac2e6a313deb86e0710ad4505cd7f7b88d563579174b8f9  
(1 row)
```

If both the original data and its encryption result are modified, the digest() function cannot identify the changes. The hmac() function can identify the changes as long as the key is not disclosed.

If the key is longer than the hash block, it will be hashed first, and the hash result will be used as the key.

Cryptographic Hash Functions

The `crypt()` and `gen_salt()` functions are used for password hashing. `crypt()` executes hashes to encrypt data, and `gen_salt()` generates salted hashes.

The algorithms in `crypt()` differ from the common MD5 and SHA1 hash algorithms in the following aspects:

- The algorithms used in `crypt()` are slow. This is the only way to make it difficult for brute-force attackers to crack passwords, which only contain a small amount of data.
- A random value (called salt) is used for encryption, so that users will get different ciphertexts even if they use the same passwords. This can protect passwords for cracking algorithms.
- The encryption results include algorithm types. Passwords can be encrypted using different algorithms for different users.
- Some of the algorithms are self-adaptive. They can slow down computing if it is too fast, and do not cause incompatibility issues with existing passwords.

The following table lists the algorithms supported by the `crypt()` function.

Table 6-7 Algorithms supported by `crypt()`

Algorithm	Maximum Password Length	Adaptability	Salt Bits	Standard Output Length	Description
bf	72	√	128	60	Blowfish-based 2a variation
md5	unlimited	×	48	34	MD5-based algorithm
xdes	8	√	24	20	Extended DES
des	8	×	12	13	Native UNIX algorithm

- `crypt()`

The syntax of `crypt()` is as follows:

```
crypt(password text, salt text) returns text
```

This function returns a hash value of the password string in `crypt(3)` format. The salt parameter is generated by the `gen_salt()` function.

For the same password, the `crypt()` function returns a different result each time, because the `gen_salt()` function generates a different salt each time. During password verification, the previously generated hash result can be used as the salt.

For example, to set a new password, run the following command:

```
UPDATE ... SET pswhash = crypt('new password', gen_salt('bf',10));
```

The hash values of the entered password and the stored password are compared.

```
SELECT (pswhash = crypt('entered password', pswhash)) AS pswmatch FROM ... ;
```

If the entered password is correct, **true** is returned.

Example:

```
create table userpwd(userid int8, pwd text);  
CREATE TABLE
```

```
insert into userpwd values (1, crypt('this is a pwd', gen_salt('bf',10)));  
INSERT 0 1
```

```
select crypt('this is a pwd', pwd)=pwd as result from userpwd where userid =1;  
result  
-----  
t  
(1 row)
```

```
select crypt('this is a wrong pwd', pwd)=pwd as result from userpwd where userid =1;  
result  
-----  
f  
(1 row)
```

- **gen_salt()**

The **gen_salt()** function is used to generate random parameters for **crypt**. The syntax is as follows:

```
gen_salt(type text [, iter_count integer ]) returns text
```

This function generates a random salt string each time. The string determines the algorithm used by the **crypt()** function. The **type** parameter specifies a hash algorithm (**des**, **xdes**, **md5**, or **bf**) for generating a string. For the **xdes** and **bf** algorithms, **iter_count** indicates the number of iterations. A large value indicates a long encryption or cracking time.

```
SELECT gen_salt('des'), gen_salt('xdes'), gen_salt('md5'), gen_salt('bf');  
gen_salt | gen_salt | gen_salt | gen_salt  
-----+-----+-----+-----  
qh      | _J9..uEUi | $1$SNgqyKAi | $2a$06$B/Etc3J8zYBV49LrDU97MO  
(1 row)
```

The salt generated by an algorithm has a fixed format. For example, in **\$2a\$06\$** in the **bf** algorithm result, **2a** indicates the 2a variation of Blowfish, and **06** indicates the number of iterations.

If **iter_count** is ignored, the default number of iterations will be used. The valid **iter_count** values depend on the algorithm used, as shown in the table below. For the **xdes** algorithm, the number of iterations must be an odd number.

Table 6-8 Iteration counts of **crypt()**

Algorithm	Default Value	Min.	Max.
xdes	725	1	16777215
bf	6	4	31

PGP Encryption Functions

The PGP encryption function of DWS complies with the OpenPGP (RFC 4880) standard, which includes requirements for symmetric key (private key) encryption and asymmetric key (public key) encryption.

An encrypted PGP message consists of the following parts:

- Session key (encrypted symmetric key or public key) of the message
- Data encrypted using the session key

For symmetric key (password) encryption:

1. The key is encrypted using the String2Key (S2K) algorithm, which is like a slowed down crypt() algorithm with a random salt. A full-length binary key will be generated.
2. If a separate session key is required, a random key will be generated. If it is not required, the S2K key will be used as the session key.
3. If the S2K key is directly used for a session, this key will be put in the session key packet. Otherwise, the S2K key will be used to encrypt the session key, and the encryption result will be put in the session key packet.

For public key encryption:

1. A random session key is generated.
2. This random key is encrypted using the public key and then put in the session key packet.

In either case, the data encryption process is as follows:

1. (Optional) Compress data, convert data to UTF-8, or convert newline characters.
2. A block consisting of random bytes is added before the data, serving as a random initial value (IV).
3. A random prefix and the SHA1 hash value suffix are added to the data.
4. The entire content is encrypted using the session key and then placed in the data packet.

Supported PGP encryption functions

- `pgp_sym_encrypt()`

Description: Encrypts a symmetric key.

Syntax:

```
pgp_sym_encrypt(data text, psu text [, options text ]) returns bytea  
pgp_sym_encrypt_bytea(data bytea, psu text [, options text ]) returns bytea
```

data indicates the data to be encrypted, **psu** indicates the PGP symmetric key, and **options** is used to set options. For details, see [Table 6-9](#).

- `pgp_sym_decrypt()`

Description: Decrypts a message encrypted using a PGP symmetric key.

Syntax:

```
pgp_sym_decrypt(msg bytea, psu text [, options text ]) returns text  
pgp_sym_decrypt_bytea(msg bytea, psu text [, options text ]) returns bytea
```

msg indicates the data to be decrypted, **psu** indicates the PGP symmetric key, and **options** is used to set options. For details, see [Table 6-9](#). To avoid generating invalid characters, you are not allowed to use the `pgp_sym_decrypt` function to decrypt bytea data. You can use the `pgp_sym_decrypt_bytea` function instead.

- `pgp_pub_encrypt()`

Description: Encrypts a public key.

Syntax:

```
pgp_pub_encrypt(data text, key bytea [, options text ]) returns bytea  
pgp_pub_encrypt_bytea(data bytea, key bytea [, options text ]) returns bytea
```

data indicates the data to be encrypted. **key** indicates the PGP public key. If a private key is used as input, an error will be returned. **options** is used to set options. For details, see [Table 6-9](#).

- **pgp_pub_decrypt()**

Description: Decrypts a message encrypted using a PGP public key.

Syntax:

```
pgp_pub_decrypt(msg bytea, key bytea [, psw text [, options text ]]) returns text  
pgp_pub_decrypt_bytea(msg bytea, key bytea [, psw text [, options text ]]) returns bytea
```

You can decrypt a message encrypted using a public key. The **key** must be the private key corresponding to the public key used for encryption. If the private key is password protected, specify the password in **psw**. If you have not specified any password but want to specify this option now, provide an empty password.

To avoid generating invalid characters, you are not allowed to use the `pgp_pub_decrypt` function to decrypt bytea data. You can use `pgp_pub_decrypt_bytea` function instead.

The **key** must be the private key corresponding to the public key used for encryption. If the private key is password protected, specify the password in **psw**. If you have not specified any password but want to specify this option now, provide an empty password. The options **parameter** is used to set options. For details, see [Table 6-9](#).

- **pgp_key_id()**

Description: Extracts the key ID of the PGP public or private key. If an encrypted message is used as the input, the ID of the key used to encrypt the message will be returned.

Syntax:

```
pgp_key_id(bytea) returns text
```

This function can return two special key IDs:

- **SYMKEY**, indicating that a message is encrypted using a symmetric key.
- **ANYKEY**, indicating that a message is encrypted using the public key, but the key ID has been deleted. To decrypt the message in this case, you need to try all the keys until you find the correct private key. `pgcrypto` does not produce such encrypted messages.

 NOTE

Different keys may have the same ID. This situation rarely occurs. In this case, the client application needs to try different keys for decryption, in the same way it deals with **ANYKEY**.

- **armor()**

Description: Converts binary data into PGP ASCII-armor format by the CRC calculation and formatting of a Base64 string.

Syntax:

```
armor(data bytea [, keys text[], values text[] ]) returns text
```

- **dearmor()**

Description: Performs the reverse conversion.

Syntax:

`dearmor(data text)` returns `bytea`

Converts the encrypted data `bytea` to the PGP ASCII-armor format, or the other way around.

data indicates the data to be converted. If multiple pairs of keys and values are specified, an armor header will be generated for each key-value pair and added to the output. The two arrays are both one-dimensional arrays with the same length, and cannot contain non-ASCII characters.

- `pgp_armor_headers()`

Description: Returns the armor header in the data.

`pgp_armor_headers(data text, key out text, value out text)` returns `setof record`

The return result is a data row set consisting of key and value columns. Any non-ASCII characters contained in the set are regarded as UTF-8 characters.

Using GnuPG to generate PGP keys

To generate a key, run the following command:

```
gpg --gen-key
```

DSA and Elgamal keys are recommended.

To use an RSA key, you must create a DSA or RSA key as the master key used only for signature, and then specify **gpg --edit-key** to add an RSA encryption subkey.

To list keys, run the following command:

```
gpg --list-secret-keys
```

To export a public key in ASCII-protected format, run the following command:

```
gpg -a --export KEYID > public.key
```

To export a private key in ASCII-protected format, run the following command:

```
gpg -a --export-secret-keys KEYID > secret.key
```

Before using these keys as the input to the PGP function, run `dearmor()` on them. Alternatively, if you can process binary data, remove **-a** from the command.

NOTICE

The PGP encryption function has the following restrictions:

- Signatures are not supported. This function does not check whether the encryption subkey belongs to the master key.
- The encryption key cannot be used as the master key. This constraint does not impose much impact, because it is rarely violated.
- Only one subkey is allowed. This may be a problem, because multiple subkeys are often required. General GPG and PGP keys cannot be used as pgcrypto encryption keys. Their usage is totally different.

PGP function parameters

The option names in the pgcrypto function are similar to those in the GnuPG function. Option values are set using equal signs (=), and the options are separated by commas (.). Example:

```
pgp_sym_encrypt(data, psw, 'compress-algo=1, cipher-algo=aes256')
```

Options other than **convert-crlf** can be used only for encryption functions. The decryption function obtains parameters from PGP data.

The most common options are **compress-algo** and **unicode-mode**. You can retain the default values for other options.

Table 6-9 pgcrypto encryption options

Option	Description	Default Value	Value	Function
cipher-algo	Cryptographic algorithm	aes128	bf, aes128, aes192, aes256, 3des, cast5	pgp_sym_encrypt, pgp_pub_encrypt
compress-algo	Compression algorithm	0	• 0: not compressed • 1: ZIP compression • 2: ZLIB compression (ZIP + Metadata + CRC)	pgp_sym_encrypt, pgp_pub_encrypt
compress-level	Compression level. A high level indicates the compression will be slow, but the data size after compression will be small. 0 disables compression.	6	0, 1-9	pgp_sym_encrypt, pgp_pub_encrypt
convert-crlf	Indicates whether to convert \n to \r\n during encryption, and whether to convert \r\n to \n during decryption. RFC4880 requires that \r\n must be used as the newline character in text data storage.	0	0, 1	pgp_sym_encrypt, pgp_pub_encrypt, pgp_sym_decrypt, pgp_pub_decrypt

Option	Description	Default Value	Value	Function
disable-mdc	SHA-1 is not used to protect data. It is used only for compatibility with old PGP products.	0	0, 1	pgp_sym_encrypt, pgp_pub_encrypt
sess-key	A separate session key is used. Public key encryption always uses a separate session key. This option is used for symmetric key encryption, which directly uses the S2K key by default.	0	0, 1	pgp_sym_encrypt
s2k-mode	S2K algorithm	3	• 0: Salt is not used. This setting is not recommended. • 1: Salt is used, but the number of iterations is fixed. • 3: Salt is used, and the number of iterations can be changed.	pgp_sym_encrypt
s2k-count	Number of iterations of the S2K algorithm	A random value between 65,536 and 253,952.	$1024 \leq \text{Value} \leq 65,011,712$	pgp_sym_encrypt and s2k-mode=3
s2k-digest-algo	Digest algorithm used during S2K calculation	sha1	md5, sha1	pgp_sym_encrypt

Option	Description	Default Value	Value	Function
s2k-cipher-algo	Password used to encrypt a separate session key	cipher-algo	bf, aes, aes128, aes192, aes256	pgp_sym_encrypt
unicode-mode	Whether to convert text data between database internal encoding and UTF-8. If the database already uses UTF-8 encoding, no conversion will be performed, but the message will be marked as UTF-8. If this parameter is not specified, the message will not be marked.	0	0, 1	pgp_sym_encrypt, pgp_pub_encrypt

Raw Encryption Functions

Raw encryption functions only run a cipher over data. They do not support any advanced functions of PGP encryption. Therefore, the following problems exist:

- They use user key directly as cipher key.
- No integrity check is performed to check whether the encrypted data was modified.
- You need to associate all encryption parameters yourself, including IV.
- Text data cannot be processed.

With the introduction of PGP encryption, these raw encryption functions are not recommended.

```
encrypt(data bytea, key bytea, type text) returns bytea
decrypt(data bytea, key bytea, type text) returns bytea
encrypt_iv(data bytea, key bytea, iv bytea, type text) returns bytea
decrypt_iv(data bytea, key bytea, iv bytea, type text) returns bytea
```

data indicates the data to be encrypted, and **type** indicates the encryption/decryption method. The syntax of the **type** parameter is as follows:

```
algorithm [ - mode ] [ /pad: padding ]
```

The options of **algorithm** are as follows:

- **bf**: Blowfish algorithm. Synonyms: **BF**, **BF-CBC**; **BLOWFISH**, **BF-CBC**; **BLOWFISH-CBC**, **BF-CBC**; **BLOWFISH-ECB**, **BF-ECB**; **BLOWFISH-CFB**, **BF-CFB**
- **aes**: AES algorithm (Rijndael-128, -192, or -256). Synonyms: **AES**, **AES-CBC**, **RIJNDAEL**, **AES-CBC**, **RIJNDAEL**, **AES-CBC**, **RIJNDAEL-CBC**, **AES-CBC**, **RIJNDAEL-ECB**, **AES-ECB**
- DES algorithm. Synonyms: **DES**, **DES-CBC**; **3DES**, **DES3-CBC**, **3DES-ECB**, **DES3-ECB**; **3DES-CBC**, **DES3-CBC**
- CAST5 algorithm. Synonym: **CAST5-CBC**

The options of **mode** are as follows:

- **cbc**: The next block depends on the previous block. (This is the default value.)
- **ecb**: Each block is encrypted separately. (This value is used only for tests.)

The options of **padding** are as follows:

- **pkcs**: The data can be of any length. (This is the default value.)
- **none**: The data must be a multiple of cipher block size.

For example, the encryption results of the following functions are the same:

```
encrypt(data, 'fooz', 'bf')
encrypt(data, 'fooz', 'bf-cbc/pad:pkcs')
```

For the **encrypt_iv** and **decrypt_iv** functions, the **iv** parameter indicates the initial value for the CBC mode. This parameter is ignored for ECB. It is truncated or padded with zeroes if not exactly block size. It defaults to all zeroes in the functions without this parameter.

Random Data Functions

- The **gen_random_bytes()** function is used to generate cryptographically strong random bytes.

gen_random_bytes(count integer) returns bytea

count indicates the number of returned bytes. The value range is 1 to 1024.

Example:

```
SELECT gen_random_bytes(16);
       gen_random_bytes
-----
\x1f1eddc11153afdde0f9e1229f8f4caf
(1 row)
```

- The **gen_random_uuid()** function is used to return a random UUID of version 4.

```
SELECT gen_random_uuid();
       gen_random_uuid
-----
2bd664a2-b760-4859-8af6-8d09ccc5b830
```

7 DWS Data Query

7.1 Single Table Query

Example table:

```
CREATE TABLE newproducts
(
  product_id INTEGER NOT NULL,
  product_name VARCHAR2(60),
  category VARCHAR2(60),
  quantity INTEGER
)
WITH (ORIENTATION = COLUMN) DISTRIBUTE BY HASH(product_id);

INSERT INTO newproducts VALUES (1502, 'earphones', 'electronics',150);
INSERT INTO newproducts VALUES (1601, 'telescope', 'toys',80);
INSERT INTO newproducts VALUES (1666, 'Frisbee', 'toys',244);
INSERT INTO newproducts VALUES (1700, 'interface', 'books',100);
INSERT INTO newproducts VALUES (2344, 'milklotion', 'skin care',320);
INSERT INTO newproducts VALUES (3577, 'dumbbell', 'sports',550);
INSERT INTO newproducts VALUES (1210, 'necklace', 'jewels', 200);
```

Simple Queries

Run the **SELECT... FROM...** statement to obtain the result from the database.

```
SELECT category FROM newproducts;
category
-----
electr
sports
jewels
toys
books
skin care
toys
(7 rows)
```

Filtering Test Results

Run the **WHERE** statement to filter the query result and find the queried part.

```
SELECT * FROM newproducts WHERE category='toys';
product_id | product_name | category | quantity
```



```
-----+-----+-----+-----  
1601 | telescope | toys | 80  
1666 | Frisbee   | toys | 244  
(2 rows)
```

Sorting Results

Use the **ORDER BY** statement to sort query results.

```
SELECT product_id,product_name,category,quantity FROM newproducts ORDER BY quantity DESC;  
product_id | product_name | category | quantity  
-----+-----+-----+-----  
3577 | dumbbell   | sports  | 550  
2344 | milklotion | skin care | 320  
1666 | Frisbee    | toys   | 244  
1210 | necklace   | jewels  | 200  
1502 | earphones  | electronics | 150  
1700 | interface  | books   | 100  
1601 | telescope  | toys   | 80  
(7 rows)
```

Limiting the Number of Query Results

If you want the query to return only part of the result, you can use the **LIMIT** statement to limit the number of records returned in the query result.

```
SELECT product_id,product_name,category,quantity FROM newproducts ORDER BY quantity DESC limit 5;  
product_id | product_name | category | quantity  
-----+-----+-----+-----  
3577 | dumbbell   | sports  | 550  
2344 | milklotion | skin care | 320  
1666 | Frisbee    | toys   | 244  
1210 | necklace   | jewels  | 200  
1502 | earphones  | electronics | 150  
(5 rows)
```

Aggregated Query

If you want query data comprehensively, you can use the **GROUP BY** statement and aggregate functions to construct an aggregated query.

```
SELECT category, string_agg(quantity,',') FROM newproducts group by category;  
category | string_agg  
-----+-----  
toys     | 80,244  
books    | 100  
sports   | 550  
jewels   | 200  
skin care | 320  
electronics | 150
```

7.2 Multi-Table Join Query

Join Types

Multiple joins are necessary for accomplishing complex queries. Joins are classified into inner joins and outer joins. Each type of joins have their subtypes.

- Inner join: inner join, cross join, and natural join.
- Outer join: left outer join, right outer join, and full join.

To better illustrate the differences between these joins, the following provides some examples.

Create the sample tables **student** and **math_score** and insert data into them. Set **enable_fast_query_shipping** to **off** (on by default), that is, the query optimizer uses the distributed framework. Set **explain_perf_mode** to **pretty** (default value) to specify the **EXPLAIN** display format.

```
CREATE TABLE student(  
  id INTEGER,  
  name varchar(50)  
);  
  
CREATE TABLE math_score(  
  id INTEGER,  
  score INTEGER  
);  
  
INSERT INTO student VALUES(1, 'Tom');  
INSERT INTO student VALUES(2, 'Lily');  
INSERT INTO student VALUES(3, 'Tina');  
INSERT INTO student VALUES(4, 'Perry');  
  
INSERT INTO math_score VALUES(1, 80);  
INSERT INTO math_score VALUES(2, 75);  
INSERT INTO math_score VALUES(4, 95);  
INSERT INTO math_score VALUES(6, NULL);  
  
SET enable_fast_query_shipping = off;  
SET explain_perf_mode = pretty;
```

Inner Join

- Inner join

Syntax:

```
left_table [INNER] JOIN right_table [ ON join_condition | USING ( join_column )]
```

Description: Rows that meet **join_condition** in both the left and right tables are joined and output. Tuples that do not meet **join_condition** are not output.

Example 1: Query students' math scores.

```
SELECT s.id, s.name, ms.score FROM student s JOIN math_score ms on s.id = ms.id;
```

```
id | name | score  
-----+-----  
2 | Lily | 75  
1 | Tom | 80  
4 | Perry | 95  
(3 rows)
```

```
EXPLAIN SELECT s.id, s.name, ms.score FROM student s JOIN math_score ms on s.id = ms.id;  
QUERY PLAN
```

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Streaming (type: GATHER)	4		13	19.47
2	-> Hash Join (3,4)	4	1MB	13	11.47
3	-> Seq Scan on math_score ms	30	1MB	8	10.10
4	-> Hash	12	16MB	9	1.28
5	-> Streaming(type: BROADCAST)	12	2MB	9	1.28
6	-> Seq Scan on student s	4	1MB	9	1.01

Predicate Information (identified by plan id)

```
2 --Hash Join (3,4)  
Hash Cond: (ms.id = s.id)
```

```
===== Query Summary =====
```

```
-----  
System available mem: 1761280KB  
Query Max mem: 1761280KB  
Query estimated mem: 4400KB  
(19 rows)
```

- Cross join

Syntax:

```
left_table CROSS JOIN right_table
```

Description: Each row in the left table is joined with each row in the right table. The number of final rows is the product of the number of rows on both sides. The product is also called Cartesian product.

Example 2: Cross join of student tables and math score tables.

```
SELECT s.id, s.name, ms.score FROM student s CROSS JOIN math_score ms;  
id | name | score
```

```
-----  
3 | Tina | 80  
2 | Lily | 80  
1 | Tom | 80  
4 | Perry | 80  
3 | Tina |  
2 | Lily |  
1 | Tom |  
4 | Perry |  
3 | Tina | 95  
2 | Lily | 95  
1 | Tom | 95  
4 | Perry | 95  
2 | Lily | 75  
3 | Tina | 75  
1 | Tom | 75  
4 | Perry | 75  
(16 rows)
```

```
EXPLAIN SELECT s.id, s.name, ms.score FROM student s CROSS JOIN math_score ms;  
QUERY PLAN
```

```
-----  
id | operation | E-rows | E-memory | E-width | E-costs  
-----  
1 | -> Streaming (type: GATHER) | 120 | | 13 | 19.89  
2 | -> Nested Loop (3,4) | 120 | 1MB | 13 | 11.89  
3 | -> Seq Scan on math_score ms | 30 | 1MB | 4 | 10.10  
4 | -> Materialize | 12 | 16MB | 9 | 1.30  
5 | -> Streaming(type: BROADCAST) | 12 | 2MB | 9 | 1.28  
6 | -> Seq Scan on student s | 4 | 1MB | 9 | 1.01
```

```
===== Query Summary =====
```

```
-----  
System available mem: 1761280KB  
Query Max mem: 1761280KB  
Query estimated mem: 4144KB  
(14 rows)
```

- Natural join

Syntax:

```
left_table NATURAL JOIN right_table
```

Description: Columns with the same name in left table and right table are joined by equi-join, and the columns with the same name are merged into one column.

Example 3: Natural join between the **student** table and the **math_score** table. The columns with the same name in the two tables are the **id** columns, therefore equivalent join is performed based on the **id** columns.

```
SELECT * FROM student s NATURAL JOIN math_score ms;
id | name | score
-----+-----+-----
1 | Tom | 80
4 | Perry | 95
2 | Lily | 75
(3 rows)
```

```
EXPLAIN SELECT * FROM student s NATURAL JOIN math_score ms;
          QUERY PLAN
-----+-----+-----+-----+-----+-----
id | operation | E-rows | E-memory | E-width | E-costs
-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 4 | 13 | 19.47
2 | -> Hash Join (3,4) | 4 | 1MB | 11.47
3 | -> Seq Scan on math_score ms | 30 | 1MB | 10.10
4 | -> Hash | 12 | 16MB | 1.28
5 | -> Streaming(type: BROADCAST) | 12 | 2MB | 1.28
6 | -> Seq Scan on student s | 4 | 1MB | 1.01
```

Predicate Information (identified by plan id)

```
-----+-----
2 --Hash Join (3,4)
   Hash Cond: (ms.id = s.id)
```

===== Query Summary =====

System available mem: 1761280KB
Query Max mem: 1761280KB
Query estimated mem: 4400KB
(19 rows)

Outer Join

- Left Join

Syntax:

```
left_table LEFT [OUTER] JOIN right_table [ ON join_condition | USING ( join_column )]
```

Description: The result set of a left outer join includes all rows of left table, not only the joined rows. If a row in the left table does not match any row in right table, the row will be **NULL** in the result set.

Example 4: Perform left join on the **student** table and **math_score** table. The right table data corresponding to the row where ID is 3 in the **student** table is filled with **NULL** in the result set.

```
SELECT s.id, s.name, ms.score FROM student s LEFT JOIN math_score ms on (s.id = ms.id);
id | name | score
-----+-----+-----
3 | Tina | 
1 | Tom | 80
2 | Lily | 75
4 | Perry | 95
(4 rows)
```

```
EXPLAIN SELECT s.id, s.name, ms.score FROM student s LEFT JOIN math_score ms on (s.id = ms.id);
          QUERY PLAN
-----+-----+-----+-----+-----+-----
id | operation | E-rows | E-memory | E-width | E-costs
-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 4 | 13 | 10.26
2 | -> Hash Left Join (3, 5) | 4 | 1MB | 2.26
3 | -> Streaming(type: REDISTRIBUTE) | 4 | 2MB | 1.11
4 | -> Seq Scan on student s | 4 | 1MB | 1.01
5 | -> Hash | 4 | 16MB | 1.11
6 | -> Streaming(type: REDISTRIBUTE) | 4 | 2MB | 1.11
7 | -> Seq Scan on math_score ms | 4 | 1MB | 1.01
```

Predicate Information (identified by plan id)

```
-----
2 --Hash Left Join (3, 5)
  Hash Cond: (s.id = ms.id)
```

===== Query Summary =====

```
-----
System available mem: 901120KB
Query Max mem: 901120KB
Query estimated mem: 7520KB
(20 rows)
```

- Right join

Syntax:

```
left_table RIGHT [OUTER] JOIN right_table [ ON join_condition | USING ( join_column )]
```

Description: Contrary to the left join, the result set of a right join includes all rows of the right table, not just the joined rows. If a row in the right table does not match any row in right table, the row will be **NULL** in the result set.

Example 5: Perform right join on the **student** table and **math_score** table. The right table data corresponding to the row where ID is 6 in the **math_score** table is filled with **NULL** in the result set.

```
SELECT ms.id, s.name, ms.score FROM student s RIGHT JOIN math_score ms on (s.id = ms.id);
id | name | score
```

```
-----+-----+-----
1 | Tom | 80
6 |    | 
4 | Perry | 95
2 | Lily | 75
```

```
EXPLAIN SELECT ms.id, s.name, ms.score FROM student s RIGHT JOIN math_score ms on (s.id = ms.id);
QUERY PLAN
```

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Streaming (type: GATHER)	30		13	19.47
2	-> Hash Left Join (3, 4)	30	1MB	13	11.47
3	-> Seq Scan on math_score ms	30	1MB	8	10.10
4	-> Hash	12	16MB	9	1.28
5	-> Streaming(type: BROADCAST)	12	2MB	9	1.28
6	-> Seq Scan on student s	4	1MB	9	1.01

Predicate Information (identified by plan id)

```
-----
2 --Hash Left Join (3, 4)
  Hash Cond: (ms.id = s.id)
```

===== Query Summary =====

```
-----
System available mem: 1761280KB
Query Max mem: 1761280KB
Query estimated mem: 5424KB
(19 rows)
```

In a right join, **Left** is displayed in the join operator. This is because a right join is actually the process replacing the left table with the right table then performing left join.

- Full join

Syntax:

```
left_table FULL [OUTER] JOIN right_table [ ON join_condition | USING ( join_column )]
```

Description: A full join is a combination of a left outer join and a right outer join. The result set of a full outer join includes all rows of the left table and the right table, not just the joined rows. If a row in the left table does not

match any row in the right table, the row will be **NULL** in the result set. If a row in the right table does not match any row in right table, the row will be **NULL** in the result set.

Example 6: Perform full outer join on the **student** table and **math_score** table. The right table data corresponding to the row where ID is 3 is filled with **NULL** in the result set. The left table data corresponding to the row where ID is 6 is filled with **NULL** in the result set.

```
SELECT s.id, s.name, ms.id, ms.score FROM student s FULL JOIN math_score ms ON (s.id = ms.id);
```

```
id | name | id | score
---+---+---+
2 | Lily | 2 | 75
4 | Perry | 4 | 95
1 | Tom | 1 | 80
3 | Tina | | 
| | 6 | 
(5 rows)
```

```
EXPLAIN SELECT s.id, s.name, ms.id, ms.score FROM student s FULL JOIN math_score ms ON (s.id = ms.id);
```

QUERY PLAN

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Streaming (type: GATHER)	30		17	20.24
2	-> Hash Full Join (3, 5)	30	1MB	17	12.24
3	-> Streaming(type: REDISTRIBUTE)	30	2MB	8	11.06
4	-> Seq Scan on math_score ms	30	1MB	8	10.10
5	-> Hash	4	16MB	9	1.11
6	-> Streaming(type: REDISTRIBUTE)	4	2MB	9	1.11
7	-> Seq Scan on student s	4	1MB	9	1.01

Predicate Information (identified by plan id)

```
2 --Hash Full Join (3, 5)
  Hash Cond: (ms.id = s.id)
```

===== Query Summary =====

```
System available mem: 1761280KB
Query Max mem: 1761280KB
Query estimated mem: 6496KB
(20 rows)
```

Differences Between the ON Condition and the WHERE Condition in Multi-Table Query

According to the preceding join syntax, except natural join and cross join, the **ON** condition (**USING** is converted to the **ON** condition during query parsing) is used on the join result of both the two tables. Generally, the **WHERE** condition is used in the query statement to restrict the query result. The **ON** join condition and **WHERE** filter condition do not contain conditions that can be pushed down to tables. The differences between **ON** and **WHERE** are as follows:

- The **ON** condition is used for joining two tables.
- **WHERE** is used to filter the result set.

To sum up, the **ON** condition is used when two tables are joined. After the join result set of two tables is generated, the **WHERE** condition is used.

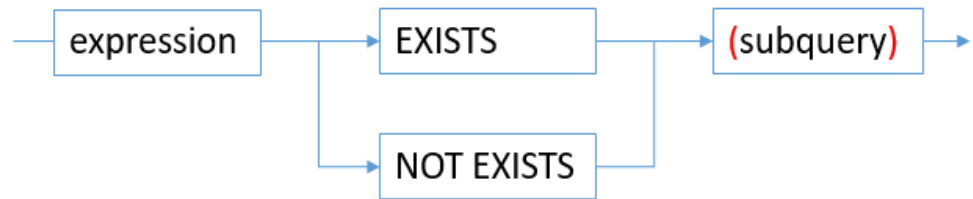
7.3 Subquery Expressions

A subquery allows you to nest one query within another, enabling more complex data query and analysis.

Subquery Expressions

- EXISTS/NOT EXISTS

Before the main query runs, the subquery runs and its result determines if the main query continues. EXISTS returns **true** if the subquery returns at least one row. **NOT EXISTS** returns **true** if the subquery returns no rows.

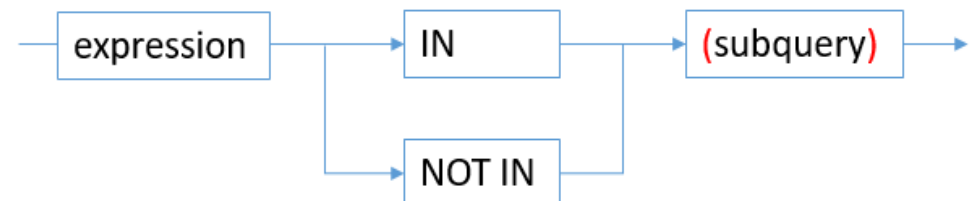


Syntax:

```
WHERE column_name EXISTS/NOT EXISTS (subquery)
```

- IN/NOT IN

IN and NOT IN are operators that check if a value is in a set of values. IN returns **true** when the outer query row matches a subquery row. **NOT IN** returns **true** when the outer query row does not match any subquery row.



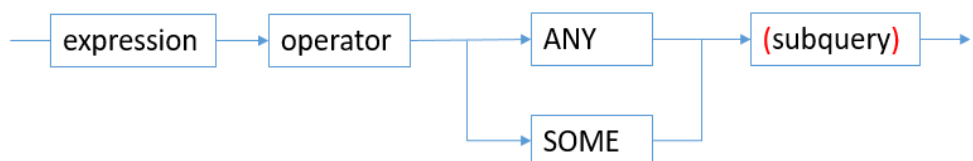
Syntax:

```
WHERE column_name IN/NOT IN (subquery)
```

- ANY/SOME

ANY indicates that any value in a subquery can match a value in an outer query. **SOME** is the same as **ANY**, but the syntax is different.

The subquery can return only one column. The expression on the left uses operators (=, <>, <, <=, >, >=) to compare the value with each subquery row. The result must be a Boolean value. The result of **ANY** is **true** if any true result is obtained. The result is **false** if no true result is found (including the case where the subquery returns no rows).



Syntax:

WHERE column_name operator ANY/SOME (subquery)

- ALL

The subquery on the right must return only one field. The expression on the left uses operators (=, <>, <, <=, >, >=) to compare the value with each subquery row. The result must be a Boolean value. The result of **ALL** is **true** if all rows yield true (including the case where the subquery returns no rows). The result is **false** if any false result is found.



Syntax:

WHERE column_name operator ALL (subquery)

Table 7-1 ALL conditions

Condition	Description
column_name > ALL(...)	The column_name value must be greater than the maximum value of a set to be true.
column_name >= ALL(...)	The column_name value must be greater than or equal to the maximum value of a set to be true.
column_name < ALL(...)	The column_name value must be smaller than the minimum value of a set to be true.
column_name <= ALL(...)	The column_name value must be smaller than or equal to the minimum value of a set to be true.
column_name <> ALL(...)	The column_name value cannot be equal to any value in a set to be true.
column_name = ALL(...)	The column_name value must be equal to any value in a set to be true.

Example

Create the **course** table and insert data into the table.

```
CREATE TABLE course(cid VARCHAR(10) COMMENT 'No.course',cname VARCHAR(10) COMMENT 'course name',teid VARCHAR(10) COMMENT 'No.teacher');  
  
INSERT INTO course VALUES('01' , 'course1' , '02');  
INSERT INTO course VALUES('02' , 'course2' , '01');  
INSERT INTO course VALUES('03' , 'course3' , '03');
```

Create the **teacher** table and insert data into the table.

```
CREATE TABLE teacher(teid VARCHAR(10) COMMENT 'Teacher ID',tname VARCHAR(10) COMMENT 'Teacher name');
```



```
INSERT INTO teacher VALUES('01', 'teacher1');
INSERT INTO teacher VALUES('02', 'teacher2');
INSERT INTO teacher VALUES('03', 'teacher3');
INSERT INTO teacher VALUES('04', 'teacher4');
```

- EXISTS/NOT EXISTS example

Query the teacher records in the **course** table.

```
SELECT * FROM teacher WHERE EXISTS (SELECT * FROM course WHERE course.teid = teacher.teid);
```

teid	tname
02	teacher2
01	teacher1
03	teacher3

(3 rows)

Query the teacher records that are not in the **course** table.

```
SELECT * FROM teacher WHERE NOT EXISTS (SELECT * FROM course WHERE course.teid = teacher.teid);
```

teid	tname
04	teacher4

(1 row)

- IN/NOT IN example

Query the **course** table for teacher information based on the teacher ID.

```
SELECT * FROM course WHERE teid IN (SELECT teid FROM teacher );
```

cid	cname	teid
01	course1	02
03	course3	03
02	course2	01

(3 rows)

Query the information about teachers who are not in the **course** table.

```
SELECT * FROM teacher WHERE teid NOT IN (SELECT teid FROM course );
```

teid	tname
04	teacher4

(1 row)

- ANY/SOME example

Compare the main query fields on the left with the subquery fields on the right to obtain the required result set.

```
SELECT * FROM course WHERE teid < ANY (SELECT teid FROM teacher where teid<>'04');
```

or

```
SELECT * FROM course WHERE teid < some (SELECT teid FROM teacher where teid<>'04');
```

cid	cname	teid
01	course1	02
02	course2	01

(2 rows)

- ALL example

The value in the **teid** column must be smaller than the minimum value in the set to be true.

```
SELECT * FROM course WHERE teid < ALL(SELECT teid FROM teacher WHERE teid<>'01');
```

```
cid | cname | teid
----+-----+----
02  | course2 | 01
(1 row)
```

Important Notes

- Duplicate subquery statements are not allowed in an SQL statement.
- Avoid scalar sub-queries whenever possible. A scalar subquery is a subquery whose result is one value and whose condition expression uses an equal operator.
- Do not use subqueries in the SELECT target columns. Otherwise, the plan cannot be pushed down, affecting the execution performance.
- It is recommended that the nested subqueries cannot exceed two layers. Subqueries cause temporary table overhead. Therefore, complex queries must be optimized based on service logic.

A subquery can be nested in the SELECT statement to implement a more complex query. A subquery can also use the results of other queries in the WHERE clause to better filter data. However, subqueries may cause query performance problems and make code difficult to read and understand. Therefore, when using SQL subqueries in DWS databases, use them based on the site requirements.

7.4 WITH Expressions

The WITH expression is used to define auxiliary statements used in large queries. These auxiliary statements are usually called common table expressions (CTE), which can be understood as a named subquery. The subquery can be referenced multiple times by its name in the query.

An auxiliary statement may use **SELECT**, **INSERT**, **UPDATE**, or **DELETE**. The **WITH** clause can be attached to a main statement, which can be a **SELECT**, **INSERT**, or **DELETE** statement.

SELECT in WITH

This section describes the usage of **SELECT** in a **WITH** clause.

Syntax

```
[WITH [RECURSIVE] with_query [, ...] ] SELECT ...
```

The syntax of **with_query** is as follows:

```
with_query_name [ ( column_name [, ...] ) ]  
AS [ [ NOT ] MATERIALIZED ] ( {select | values | insert | update | delete} )
```

 CAUTION

- If you use **MATERIALIZED**, the subquery runs once and its result set is saved. If you use **NOT MATERIALIZED**, the subquery is replaced with its reference in the main query.
- The SQL statement specified by the AS statement of a CTE must be a statement that can return query results. It can be a common **SELECT** query statement or other data modification statements such as **INSERT**, **UPDATE**, **DELETE**, and **VALUES**. When using a data modification statement, you need to use the **RETURNING** clause to return tuples. Example:

```
WITH s AS (INSERT INTO t VALUES(1) RETURNING a) SELECT * FROM s;
```
- A **WITH** expression indicates the CTE definition in a SQL statement block. Multiple CTEs can be defined at the same time. You can specify column names for each CTE or use the aliases of the columns in the query output. Example:

```
WITH s1(a, b) AS (SELECT x, y FROM t1), s2 AS (SELECT x, y FROM t2) SELECT * FROM s1 JOIN s2 ON s1.a=s2.x;
```

This statement defines two CTEs: **s1** and **s2**. **s1** specifies the column names **a** and **b**, and **s2** does not specify the column names. Therefore, the column names are the output column names **x** and **y**.
- Each CTE can be referenced zero, one, or more times in the main query.
- CTEs with the same name cannot exist in the same statement block. If CTEs with the same name exist in different statement blocks, the CTE in the nearest statement block is referenced.
- An SQL statement may contain multiple SQL statement blocks. Each statement block can contain a **WITH** expression. The CTE in each **WITH** expression can be referenced in the current statement block, subsequent CTEs of the current statement block, and sub-layer statement blocks, however, it cannot be referenced in the parent statement block. The definition of each CTE is also a statement block. Therefore, a **WITH** expression can also be defined in the statement block.
- When the CTE is materialized because **MATERIALIZED** is specified or the CTE is referenced for multiple times, the CTE is regarded as a table. The query or filter condition in the CTE cannot be independently elevated outside the CTE for other optimization. In the following example, the filter condition in the CTE cannot be deduced equivalently with the JOIN condition. As a result, the partitioned tables involved in JOIN cannot be optimized using partition pruning.

```
CREATE TABLE date_part(ds date) distribute by hash(ds);
```

```
CREATE TABLE t1(id1 int, ds date) distribute by hash(ds)
partition by range(ds)
( partition p1 values less than('20250101'),
  partition p2 values less than('20250201'),
  partition p3 values less than('20250301'),
  partition p4 values less than('20250401'),
  partition p5 values less than('20250501'),
  partition p6 values less than('20250601'));
```

```
CREATE TABLE t3(id3 int, ds date) distribute by hash(ds)
partition by range(ds)
( partition p1 values less than('20250101'),
  partition p2 values less than('20250201'),
  partition p3 values less than('20250301'),
  partition p4 values less than('20250401'),
  partition p5 values less than('20250501'),
  partition p6 values less than('20250601'));
```

```
explain verbose -- When the CTE is materialized, the query optimizer cannot elevate the filter
conditions into the JOIN, thus hindering partition pruning on the partitioned tables involved in the JOIN.
with dt as (select ds from date_part where ds>='2025-05-04' and ds < '2025-05-06'),
t1 as(select id1 from t1 join dt t2 on t1.ds = t2.ds),
t2 as (select id3 from t3 join dt t2 on t3.ds = t2.ds) select * from t1 join t2 on t1.id1 = 100;
```

```
explain with dt as (select ds from date_part where ds>='2025-05-04' and ds < '2025-05-06'),
t1 as(select id1 from t1 join dt t2 on t1.ds = t2.ds),
t2 as(select id3 from t3 join dt t2 on t3.ds = t2.ds)
select * from t1 join t2 on t1.id1 = 100;
```

QUERY PLAN

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	2		8	66.27
2	-> Vector Streaming (type: GATHER)		2		8 66.27
3	-> Vector Nest Loop (4,15)	2	1MB	8	60.27
4	-> Vector CTE Append(5, 7)		2 1MB	12	40.26
5	-> Vector Streaming(type: LOCAL GATHER)		1 2MB		8 20.01
6	-> CStore Scan on date_part [5, CTE dt(0)]	1	1MB		8 20.01
7	-> Vector Nest Loop (8,9)	2	1MB	12	20.25
8	-> Vector CTE Scan on dt(0) t2	1	1MB		8 0.02
9	-> Vector Materialize	4	16MB	4	20.22
10	-> Vector Streaming(type: BROADCAST)		4 2MB		4 20.21
11	-> Vector Nest Loop (12,14)	2	1MB	4	20.04
12	-> Vector Partition Iterator	1	1MB	12	20.00
13	-> Partitioned CStore Scan on t1	1	1MB	12	20.00
14	-> Vector CTE Scan on dt(0) t2	1	1MB		8 0.02
15	-> Vector Partition Iterator	1	1MB	12	20.00
16	-> Partitioned CStore Scan on t3	1	1MB	12	20.00

Predicate Information (identified by plan id)

```
3 --Vector Nest Loop (4,15)
  Join Filter: (t2.ds = t3.ds)
6 --CStore Scan on date_part
  CU Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds <
'2025-05-06 00:00:00'::timestamp without time zone))
  Pushdown Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds
< '2025-05-06 00:00:00'::timestamp without time zone))
11 --Vector Nest Loop (12,14)
  Join Filter: (public.t1.ds = t2.ds)
12 --Vector Partition Iterator
  Iterations: 6
13 --Partitioned CStore Scan on t1
  CU Predicate Filter: (id1 = 100)
  Pushdown Predicate Filter: (id1 = 100)
Partitions Selected by Static Prune: 1..6
15 --Vector Partition Iterator
  Iterations: 6
16 --Partitioned CStore Scan on t3
Partitions Selected by Static Prune: 1..6
```

===== Query Summary =====

```
-----
System available mem: 2990080KB
Query Max mem: 2990080KB
Query estimated mem: 11312KB
Turbo Engine: true
(45 rows)
```

```
explain -- When the CTE is not materialized, the database expands it into a subquery in the query plan.
In this expansion, the optimizer can integrate the CTE's filter conditions with the JOIN conditions from
the outer query. This integration allows for partition pruning to be applied to partitioned tables
involved in the JOIN.
```

with dt as not materialized

```
(select ds from date_part where ds>='2025-05-04' and ds < '2025-05-06'),
t1 as(select id1 from t1 join dt t2 on t1.ds = t2.ds),
t2 as(select id3 from t3 join dt t2 on t3.ds = t2.ds) select * from t1 join t2 on t1.id1 = 100;
```

QUERY PLAN

id	operation	E-rows	E-memory	E-width	E-costs
1	-> Row Adapter	2		8	52.26
2	-> Vector Streaming (type: GATHER)		2		8 52.26
3	-> Vector Nest Loop (4,6)	2	1MB	8	46.26
4	-> Vector Streaming(type: BROADCAST)		2	2MB	8 20.10
5	-> CStore Scan on date_part	1	1MB		8 20.01
6	-> Vector Materialize	2	16MB	16	26.15
7	-> Vector Nest Loop (8,11)	2	1MB	16	26.14
8	-> Vector Streaming(type: BROADCAST)		2	2MB	12 3.10
9	-> Vector Partition Iterator	1	1MB	12	3.01
10	-> Partitioned CStore Scan on t3	1	1MB		12 3.01
11	-> Vector Nest Loop (12,14)	2	1MB	4	23.03
12	-> Vector Partition Iterator	1	1MB	12	3.01
13	-> Partitioned CStore Scan on t1	1	1MB		12 3.01
14	-> CStore Scan on date_part	1	1MB	8	20.01

Predicate Information (identified by plan id)

```
3 --Vector Nest Loop (4,6)
  Join Filter: (t3.ds = public.date_part.ds)
5 --CStore Scan on date_part
  CU Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds <
'2025-05-06 00:00:00'::timestamp without time zone))
  Pushdown Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds
< '2025-05-06 00:00:00'::timestamp without time zone))
9 --Vector Partition Iterator
  Iterations: 1
10 --Partitioned CStore Scan on t3
  CU Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds <
'2025-05-06 00:00:00'::timestamp without time zone))
  Pushdown Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds
< '2025-05-06 00:00:00'::timestamp without time zone))
Partitions Selected by Static Prune: 6
11 --Vector Nest Loop (12,14)
  Join Filter: (public.t1.ds = public.date_part.ds)
12 --Vector Partition Iterator
  Iterations: 1
13 --Partitioned CStore Scan on t1
  CU Predicate Filter: ((id1 = 100) AND (ds >= '2025-05-04 00:00:00'::timestamp without time zone)
AND (ds < '2025-05-06 00:00:00'::timestamp without time zone))
  Pushdown Predicate Filter: ((id1 = 100) AND (ds >= '2025-05-04 00:00:00'::timestamp without
time zone) AND (ds < '2025-05-06 00:00:00'::timestamp without time zone))
Partitions Selected by Static Prune: 6
14 --CStore Scan on date_part
  CU Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds <
'2025-05-06 00:00:00'::timestamp without time zone))
  Pushdown Predicate Filter: ((ds >= '2025-05-04 00:00:00'::timestamp without time zone) AND (ds
< '2025-05-06 00:00:00'::timestamp without time zone))
```

===== Query Summary =====

```
System available mem: 2990080KB
Query Max mem: 2990080KB
Query estimated mem: 12352KB
Turbo Engine: true
(48 rows)
```

The purpose of **SELECT** in **WITH** is to break down complex queries into simple parts. Example:

```
WITH regional_sales AS (  
    SELECT region, SUM(amount) AS total_sales  
    FROM orders  
    GROUP BY region  
) , top_regions AS (  
    SELECT region  
    FROM regional_sales  
    WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)  
)  
SELECT region,  
    product,  
    SUM(quantity) AS product_units,  
    SUM(amount) AS product_sales  
FROM orders  
WHERE region IN (SELECT region FROM top_regions)  
GROUP BY region, product;
```

The **WITH** clause defines two auxiliary statements: **regional_sales** and **top_regions**. The output of **regional_sales** is used in **top_regions**, and the output of **top_regions** is used in the main **SELECT** query. This example can be written without **WITH**. In that case, it must be written with a two-layer nested sub-**SELECT** statement, making the query longer and difficult to maintain.

Recursive WITH Query

By declaring the keyword **RECURSIVE**, a **WITH** query can reference its own output.

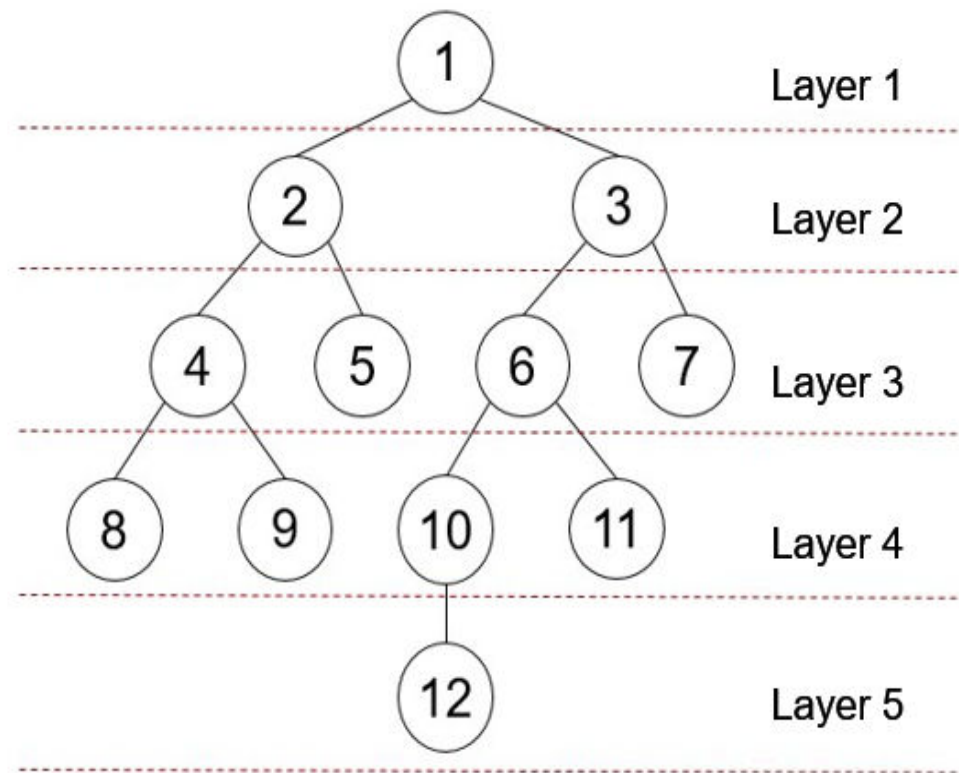
The common form of a recursive **WITH** query is as follows:

```
non_recursive_term UNION [ALL] recursive_term
```

UNION performs deduplication when combining sets. **UNION ALL** directly combines result sets without deduplication. Only recursive items can contain references to the query output.

When using recursive **WITH**, ensure that the recursive item of the query does not return a tuple. Otherwise, the query will loop infinitely.

The table **tree** is used to store information about all nodes in the following figure.



The table definition statement is as follows:

```
CREATE TABLE tree(id INT, parentid INT);
```

The data in the table is as follows:

```
INSERT INTO tree VALUES(1,0),(2,1),(3,1),(4,2),(5,2),(6,3),(7,3),(8,4),(9,4),(10,6),(11,6),(12,10);
```

```
SELECT * FROM tree;
```

```
id | parentid
```

```
---+-----
```

```
1 | 0
```

```
2 | 1
```

```
3 | 1
```

```
4 | 2
```

```
5 | 2
```

```
6 | 3
```

```
7 | 3
```

```
8 | 4
```

```
9 | 4
```

```
10 | 6
```

```
11 | 6
```

```
12 | 10
```

```
(12 rows)
```

You can run the following **WITH RECURSIVE** statement to return the nodes and hierarchy information of the entire tree starting from node 1 at the top layer:

```
WITH RECURSIVE nodeset AS
```

```
(
```

```
-- recursive initializing query
```

```
SELECT id, parentid, 1 AS level FROM tree
```

```
WHERE id = 1
```

```
UNION ALL
```

```
-- recursive join query
```

```
SELECT tree.id, tree.parentid, level + 1 FROM tree, nodeset
```

```
WHERE tree.parentid = nodeset.id
)
SELECT * FROM nodeset ORDER BY id;
```

In the preceding query, a typical **WITH RECURSIVE** expression contains the CTE of at least one recursive query. The CTE is defined as a **UNION ALL** set operation. The first branch is the recursive start query, and the second branch is the recursive join query, the first part is referenced for continuous recursive join. When this statement is executed, the recursive start query is executed once, and the join query is executed several times. The results are added to the start query result set until the results of some join queries are empty.

The command output is as follows:

id	parentid	level
1	0	1
2	1	2
3	1	2
4	2	3
5	2	3
6	3	3
7	3	3
8	4	4
9	4	4
10	6	4
11	6	4
12	10	5

(12 rows)

According to the returned result, the start query result contains the result set whose level is 1. The join query is executed for five times. The result sets whose levels are 2, 3, 4, and 5 are output for the first four times. During the fifth execution, there is no record whose parentid is the same as the output result set ID, that is, there is no redundant child node. Therefore, the query ends.

NOTE

DWS supports distributed execution of **WITH RECURSIVE** expressions. **WITH RECURSIVE** involves cyclic data. DWS introduces the **max_recursive_times** parameter to control the maximum number of cycles in **WITH RECURSIVE**. The default value is **200**. If the number of cycles exceeds **200**, an error is reported.

Data Modification Statements in WITH

Use the **INSERT**, **UPDATE**, and **DELETE** commands in the **WITH** clause. This allows the user to perform multiple different operations in the same query. The following is an example:

```
WITH moved_tree AS (
  DELETE FROM tree
  WHERE parentid = 4
  RETURNING *)
INSERT INTO tree_log
SELECT * FROM moved_tree;
```

The preceding query example actually moves rows from **tree** to **tree_log**. The **DELETE** command in the **WITH** clause deletes the specified rows from **tree**, returns their contents through the **RETURNING** clause, and then the main query reads the output and inserts it into **tree_log**.

To retrieve the modified content instead of the target table, the data modification statement in the **WITH** clause should include the **RETURNING** clause. This clause

creates a temporary table that can be accessed by the rest of the query. If a data modification statement in the **WITH** statement lacks a **RETURNING** clause, it cannot form a temporary table and cannot be referenced in the remaining queries.

If the **RECURSIVE** keyword is specified, recursive self-reference is not allowed in data modification statements. In some cases, you can bypass this restriction by referencing the output of recursive the **WITH** statement. For example:

```
WITH RECURSIVE included_parts(sub_part, part) AS (  
    SELECT sub_part, part FROM parts WHERE part = 'our_product'  
    UNION ALL  
    SELECT p.sub_part, p.part  
    FROM included_parts pr, parts p  
    WHERE p.part = pr.sub_part  
)  
DELETE FROM parts  
WHERE part IN (SELECT part FROM included_parts);
```

This query will remove all direct or indirect subparts of a product.

The substatements in the **WITH** clause are executed at the same time as the main query. Therefore, when using the data modification statement in a **WITH** statement, the actual update order is in an unpredictable manner. All statements are executed in the same snapshot, and the effect of the statements is invisible on the target table. This mitigates the unpredictability of the actual order of row updates and means that **RETURNING** data is the only way to convey changes between different **WITH** substatements and the main query.

In this example, the outer layer **SELECT** can return the data before the update.

```
WITH t AS (  
    UPDATE tree SET id = id + 1  
    RETURNING *)  
SELECT * FROM tree;
```

In this example, the external **SELECT** returns the updated data.

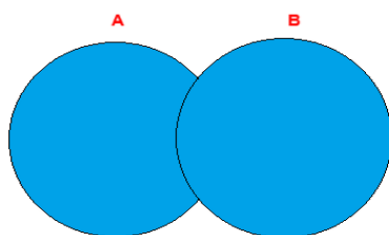
```
WITH t AS (  
    UPDATE tree SET id = id + 1  
    RETURNING *)  
SELECT * FROM t;
```

The same row cannot be updated twice in a single statement. Otherwise, the update effect will be unpredictable. If only one update takes effect, it is difficult (and sometimes impossible) to predict which one takes effect.

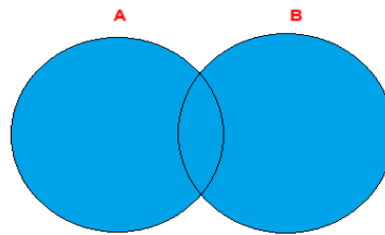
7.5 UNION Operators

UNION is a powerful SQL operator that combines the result sets of two or more **SELECT** statements into one. During combination, the number of columns and data types in the two tables must be the same and correspond to each other. Use the **UNION** or **UNION ALL** keyword between **SELECT** statements.

UNION removes duplicate rows, while **UNION ALL** keeps them. Deduplication is time-consuming, so **UNION ALL** can be faster than **UNION** if the data sets are already distinct by logic.



The UNION operator combines the results of two queries and removes any duplicates.



The UNION ALL operator combines the results of two queries and keep all the duplicates.

Syntax

```
SELECT column,... FROM table1 UNION [ALL]SELECT column,... FROM table2
```

Example

Step 1 Create the student information table **student** (ID, name, gender, and school).

```
SET current_schema=public;  
DROP TABLE IF EXISTS student;  
CREATE table student(  
sld VARCHAR(10) NOT NULL,  
sname VARCHAR(10) NOT NULL,  
sgender VARCHAR(10) NOT NULL,  
sschool VARCHAR(10) NOT NULL);
```

Step 2 Insert data into the **student** table.

```
INSERT INTO student VALUES('s01' , 'ZhaoLei' , 'male' , 'NENU');  
INSERT INTO student VALUES('s02' , 'QianDian' , 'male' , 'SJTU');  
INSERT INTO student VALUES('s03' , 'SunFeng' , 'male' , 'Tongji');  
INSERT INTO student VALUES('s04' , 'LiYun' , 'male' , 'CCOM');  
INSERT INTO student VALUES('s05' , 'ZhouMei' , 'female' , 'FuDan');  
INSERT INTO student VALUES('s06' , 'WuLan' , 'female' , 'WHU');  
INSERT INTO student VALUES('s07' , 'ZhengZhu' , 'female' , 'NWAUFU');  
INSERT INTO student VALUES('s08' , 'ZhangShan' , 'female' , 'Tongji');
```

Step 3 View the student table.

```
SELECT * FROM student;
```

Information similar to the following is displayed.

sld	sname	sgender	sschool
s02	QianDian	male	SJTU
s05	ZhouMei	female	FuDan
s08	ZhangShan	female	Tongji
s03	SunFeng	male	Tongji
s06	WuLan	female	WHU
s01	ZhaoLei	male	NENU
s04	LiYun	male	CCOM
s07	ZhengZhu	female	NWAUFU

(8 rows)

Step 4 Create the teacher information table **teacher** (ID, name, gender, and school).

```
DROP TABLE IF EXISTS teacher;  
CREATE table teacher(  
tid VARCHAR(10) NOT NULL,  
tname VARCHAR(10) NOT NULL,  
tgender VARCHAR(10) NOT NULL,  
tschool VARCHAR(10) NOT NULL);
```

Step 5 Insert data to the **teacher** table.

```
INSERT INTO teacher VALUES('t01' , 'ZhangLei', 'male', 'FuDan');  
INSERT INTO teacher VALUES('t02' , 'LiLiang', 'male', 'WHU');  
INSERT INTO teacher VALUES('t03' , 'WangGang', 'male', 'Tongji');
```

Step 6 Query the **teacher** table.

```
SELECT * FROM teacher;
```

tid	tname	tgender	tschool
t03	WangGang	male	Tongji
t02	LiLiang	male	WHU
t01	ZhangLei	male	FuDan

(3 rows)

Step 7 Use **UNION** (combine and deduplicate) to obtain the schools of students and teachers and sort the schools in ascending order by initial letter of the school name.

```
SELECT t.school FROM (  
    SELECT sschool AS school  
    FROM student  
    UNION  
    SELECT tschool AS school  
    FROM teacher  
) t  
ORDER BY t.school ASC;
```

Information similar to the following is displayed.

school
CCOM
FuDan
NENU
NWAFU
SJTU
Tongji
WHU

(7 rows)

Step 8 Use **UNION ALL** (combine without deduplication) to obtain the schools of all students and teachers and sort the schools by initial letter of the school name in ascending order.

```
SELECT t.school FROM (  
    SELECT sschool AS school  
    FROM student  
    UNION ALL  
    SELECT tschool AS school  
    FROM teacher  
) t  
ORDER BY t.school ASC;
```

```
school
-----
CCOM
FuDan
FuDan
NENU
NWAUFU
SJTU
Tongji
Tongji
Tongji
WHU
WHU
(11 rows)
```

Step 9 Use **UNION ALL** (combine the result sets of SQL statements with **WHERE** clause) to get all information about students and teachers from "Tongji" and sort by student and teacher number in ascending order.

```
SELECT t.* FROM (
  SELECT Sid AS id,Sname AS name,Sgender AS gender,Sschool AS school
  FROM student
  WHERE Sschool='Tongji'
  UNION ALL
  SELECT Tid AS id,Tname AS name,Tgender AS gender,Tschool AS school
  FROM teacher
  WHERE Tschool='Tongji'
) t
ORDER BY t.id ASC;
```

----End

Summary

In actual service scenarios, pay attention to the following points when using **UNION** and **UNION ALL**:

- The number of SQL fields and field types on the left and right sides must be the same.
- Check whether data deduplication (deduplication before combination or during combination) is needed based on service requirements.
- Based on the data volume, evaluate the SQL execution efficiency and determine whether to use temporary tables.
- Select **UNION** or **UNION ALL** wisely and consider the complexity when writing SQL statements.

7.6 SQL on Hudi

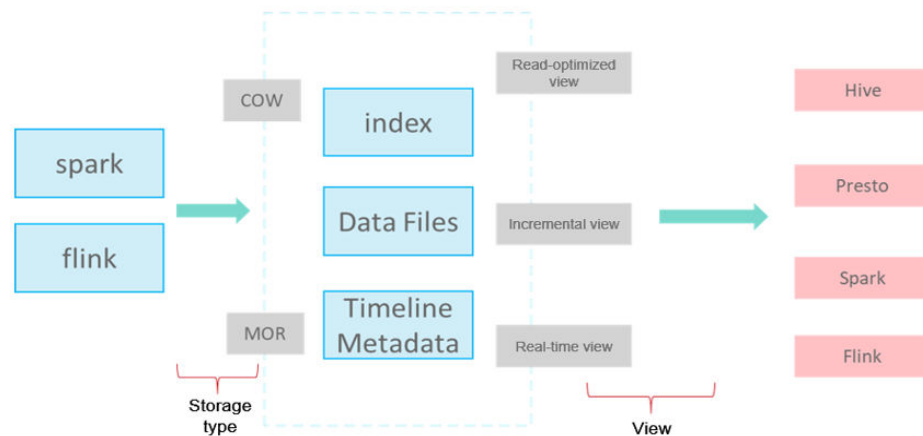
7.6.1 Introduction to Hudi

SQL on Hudi of DWS is supported only in 9.1.0.100 and later versions.

Apache Hudi indicates Hadoop Upserts Deletes and Incrementals. It is used to manage large analytical datasets stored in the distributed file system (DFS) of the Hadoop big data system.

Hudi is not just a data format. It is also a set of data access methods (similar to the access layer of DWS storage). In Apache Hudi 0.9, big data components such

as Spark and Flink have their own clients. The following figure shows the logical storage of Hudi.



- Write Mode

COW: copy-on-write, which is suitable for scenarios where updates are rare.

MOR: replication on read. For UPDATE & DELETE, delta log files are written incrementally. During analysis, base and delta log files are compacted asynchronously.

- Storage Format

index: index of the primary key. The default value is bloomfilter at the file group level.

data files: base file + delta log file (for updating and deleting base files)

timeline metadata: manages version logs.

- Views

Read-optimized view: reads the base file generated after compaction. The reading of data that is not compacted has some latency (efficient read).

Real-time view: reads the latest data. The base file and delta file are combined during the read (frequent updates).

Incremental view: reads the incremental data written to Hudi, similar to CDC (stream and batch integration).

7.6.2 Preparations Before Using Hudi

Prerequisites

- You have created an OBS agency and OBS data source. For details, see section "Managing OBS Data Sources" in the *Data Warehouse Service User Guide*.
- Before using SQL on Hudi, ensure that the **enable_dws_bigdata** parameter is enabled. If an error message similar to "flight into error" is displayed, contact technical support to enable this parameter.

Authorizing the Use of Data Sources

Run the **GRANT** command to grant a user the permission to use data sources.

```
GRANT USAGE ON FOREIGN SERVER server_name TO role_name;
```

Example:

Grant the user **u1** the permission to access the data source **hudi_server**.

```
GRANT USAGE ON FOREIGN SERVER hudi_server TO u1;
```

Granting Permissions for Using Foreign Tables

Run the following command to grant a user the permission to use foreign tables:

```
ALTER USER role_name USEFT;
```

Example:

Grant the user **u1** the permission to access foreign tables.

```
ALTER USER u1 USEFT;
```

7.6.3 Hudi User Interfaces

Querying Real-Time Views and Incremental Views

DWS provides table-level parameters similar to spark-sql to support real-time and incremental views.

The parameters are described as follows. Replace **SCHEMA.FOREIGN_TABLE** with the actual schema name and foreign table name.

Table 7-2 Parameters for querying real-time views and incremental views

Parameter	Value	Description
hoodie.SCHEMA.FOREIGN_TABLE .consume.mode	SNAPSHOT	Queries the real-time view.
	INCREMENTAL	Queries the incremental view.
hoodie.SCHEMA.FOREIGN_TABLE .consume. start.timestamp	hudi timestamp	Specifies the start commit of incremental synchronization.
hoodie.SCHEMA.FOREIGN_TABLE .consume. ending.timestamp	hudi timestamp	Specifies the end commit of incremental synchronization. If this parameter is not specified, the latest commit is used.

 NOTE

- The preceding parameters can be set by running the **set** command and are valid only in the current session. You can run the **reset** command to restore the default values.
- You can use the system function **pg_catalog.pg_show_custom_settings()** to query the parameter setting details.
- When querying the incremental views of the **MOR** tables, you need to use the **WHERE** conditions to filter the **_hoodie_commit_time** field to prevent the log file data that is not combined and does not meet the conditions from being read. This operation is not required for **COW** tables.

Querying Hudi Foreign Table and Automatically Synchronizing Tasks

DWS provides a series of system functions to obtain Hudi foreign table information and create Hudi automatic synchronization tasks. The automatic Hudi synchronization task periodically synchronizes data from Hudi foreign tables to DWS internal tables.

Table 7-3 Hudi system functions

Function	Type	Functionality
pg_show_custom_settings()	Built-in functions	Queries details about the parameter settings of an HUDI foreign table.
hudi_get_options(regclass)	Built-in functions	Queries the attributes of an HUDI foreign table (hoodie.properties).
hudi_get_max_commit(regclass)	Built-in functions	Obtains the latest commit timestamp of the current HUDI foreign table.
hudi_sync_task_submit(regclass, regclass)	Built-in functions	Submits the HUDI automatic synchronization task.
hudi_sync_task_submit(regclass, regclass, text, text)		
hudi_show_sync_state()	Built-in functions	Obtains the synchronization status of the HUDI automatic synchronization task.
hudi_sync(regclass, regclass)	Stored procedure	Specifies the entry for invoking the HUDI automatic synchronization task.
hudi_sync_custom(regclass, regclass, text)	Stored procedure	Specifies the entry for invoking the HUDI automatic synchronization task. Users can define the mapping between fields in the target table and data source table.
hudi_set_sync_commit(regclass, regclass, text)	Built-in functions	Sets the start timestamp of the first synchronization of the HUDI automatic synchronization task to prevent resynchronization.

Function	Type	Functionality
hoodie_set_sync_commit(text, text)		Sets the start timestamp of the next synchronization of a HUDI automatic synchronization task. You can use it to sync historical data again or to skip some data.

7.6.4 Creating Hudi Data (Foreign Table)

DWS accesses Hudi data on OBS through foreign tables. Compared with common foreign tables, Hudi foreign tables do not require additional parameter settings. You only need to set **format** to **hoodie** to specify the format of the data source file. For details, see "CREATE FOREIGN TABLE (SQL on OBS or Hadoop)" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

Obtaining the Definitions of Tables on MRS.

DWS Hudi foreign tables are read-only. You need to specify the columns and column types of the target data before creating a foreign table.

For example, for a Hudi table on MRS, you can use spark-sql to query the original table definitions:

```
SHOW create table rtd_mfdd_int_currency_t;
```

Compiling DWS Table Definitions

Copy the definitions of all columns in the MRS table, create a Hudi foreign table, and perform appropriate type conversion to be compatible with DWS syntax.

```
CREATE FOREIGN TABLE rtd_mfdd_int_currency_ft(  
  _hoodie_commit_time text,  
  _hoodie_commit_seqno text,  
  _hoodie_record_key text,  
  _hoodie_partition_path text,  
  _hoodie_file_name text,  
  ...  
)SERVER hudi_server OPTIONS (  
  foldername '/erpgc-test-01/s000/u1/rtd_mfdd_int_currency_t/',  
  format 'hoodie',  
  encoding 'utf-8'  
)distribute by roundrobin;
```

foldername indicates the storage path of the Hudi data on OBS, which corresponds to **LOCATION** in the Spark-sql table definitions of MRS. The path must end with a slash (/).

7.6.5 Synchronizing Hudi Tasks

Creating a Hudi Task

Migration

If data has been imported to the DWS table using CDL, use SQL on Hudi to migrate data. Alternatively, use CDM to perform full initialization and then use SQL on Hudi to synchronize incremental data.

Step 1 To create the **hudi.hudi_sync_state** synchronization status table, you must have the administrator permission.

```
SELECT pg_catalog.create_hudi_sync_table();
```

Generally, hudi.hudi_sync_state is created only once in each database.

Step 2 To set the synchronization progress, you must have the INSERT and UPDATE permissions on the target table and the SELECT permission on the Hudi foreign table. Otherwise, the synchronization progress cannot be set.

```
SELECT hudi_set_sync_commit('SCHEMA.TABLE', 'SCHEMA.FOREIGN_TABLE', 'LATEST_COMMIT');
```

Where:

- **SCHEMA.TABLE** indicates the name of the table to which data is synchronized. **schema** indicates the schema name.
- **SCHEMA.FOREIGN_TABLE** indicates the name of the OBS foreign table. **schema** indicates the schema name.
- **LATEST_COMMIT** indicates the end time of the Hudi synchronization.

Example: Data has been synchronized to the target table **public.in_rel** from Hudi by **20220913152131**. Use SQL on Hudi to continue to export data from the OBS foreign table **hudi_read1**.

```
SELECT hudi_set_sync_commit('public.in_rel', 'public.hudi_read1', '20220913152131');
```

Step 3 Submit the Hudi synchronization task.

```
SELECT hudi_sync_task_submit('SCHEMA.TABLE', 'SCHEMA.FOREIGN_TABLE');
```

Example: Use SQL on Hudi to continue to export data from the OBS foreign table **hudi_read1** to the target table **public.in_rel**.

```
SELECT hudi_sync_task_submit('public.in_rel', 'public.hudi_read1');
```

----End

Creation

If the DWS table is empty and data is synchronized from Hudi for the first time, run the following command to create a task:

```
SELECT hudi_sync_task_submit('SCHEMA.TABLE', 'SCHEMA.FOREIGN_TABLE');
```

Querying Hudi Synchronization Tasks

Querying Hudi Synchronization Tasks

Syntax:

```
SELECT * FROM pg_task_show('SQLonHudi');
```

task_id in the query result is the unique ID of the Hudi synchronization task.

Example:

```
SELECT * FROM pg_task_show('SQLonHudi');
task_id      |
what         | category_id | userid | is_broken |
```

interval	time_cons	start_time	end_time	parameter	task_rank	next_start_time
	next_end_time	last_log	failure_times			

+-----+-----+-----+-----+-----+-----+-----						
+-----+-----+-----+-----+-----+-----+-----						
+-----+-----+-----+-----+-----+-----+-----						
64d257e9-1e9b-0d00-3ce3-7e61b5e0ffe call pg_catalog.hudi_sync_custom('public.hudi_read_target',						
'public.hudi_read101', '{"_hoodie_commit_seqno": "_hoodie_commit_seqno", "id": "id", "ts": "ts",						
"long_field": "ts"}'); SQLonHudi 10 f '00:00:10' 2023-08-08 22:58:15.846903+08						
5 2023-08-08 22:58:15.846903+08 2023-08-08 22:58:24.846903+08						
0						
(1 row)						

Suspending Hudi Synchronization Tasks

Suspend a Hudi synchronization task by specified **task_id**.

Syntax:

```
SELECT pg_task_pause('task_id');
```

Example:

Suspend the synchronization task whose **task_id** is **64479410-a04c-0700-d150-3037d700ffe**.

```
SELECT pg_task_pause('64479410-a04c-0700-d150-3037d700ffe');
```

Resuming Hudi Synchronization Tasks

Resume a Hudi synchronization task by specifying **task_id**.

Syntax:

```
SELECT pg_task_resume('task_id');
```

Example:

Resume the synchronization task whose **task_id** is **64479410-a04c-0700-d150-3037d700ffe**.

```
SELECT pg_task_resume('64479410-a04c-0700-d150-3037d700ffe');
```

Resetting a Hudi Synchronization Task with Consecutive Failures

Reset a Hudi synchronization task with consecutive failures by specifying **task_id**.

If the number of consecutive failures is greater than or equal to 10, the task is automatically suspended. You need to manually call the **pg_task_resume()** function to reset the task.

```
SELECT pg_task_resume('task_id');
```

Deleting a Hudi Synchronization Task

Delete a Hudi synchronization task by specifying **task_id**.

Syntax:

```
SELECT pg_task_remove('task_id');
```

Example:

Delete the synchronization task whose **task_id** is **64479410-a04c-0700-d150-3037d700fffe**.

```
SELECT pg_task_remove('64479410-a04c-0700-d150-3037d700fffe');
```

Querying Historical Hudi Synchronization Task Information

Use the **hudi_sync_state_history_view** view to query information about past Hudi synchronization tasks. This view is supported only by clusters of version 9.1.0 and later.

```
SELECT * FROM pg_catalog.hudi_sync_state_history_view;
```

Table 7-4 hudi_sync_state_history_view columns

Column	Type	Description
task_id	text	Task ID
target_tbl	text	Name of the synchronization target table
source_ftbl	text	Name of the synchronization source table (foreign table)
latest_commit	text	Timestamp of the latest successful synchronization
latest_sync_count	bigint	Number of rows that are successfully synchronized last time
latest_sync_start	timestamp with time zone	Start time of the latest synchronization task
latest_sync_end	timestamp with time zone	Time when the latest synchronization task ends
hudi_flushdisk_time	text	Time when the hudi file is flushed to disks

Querying the Status of a Hudi Synchronization Task

Use the **hudi_show_sync_state()** function to query the status of a Hudi synchronization task.

```
SELECT * FROM hudi_show_sync_state();
```

7.6.6 Querying a Hudi Foreign Table

You can query data in a Hudi foreign table. By default, it gives you a real-time view. You can set parameters to query the incremental data.

Querying Incremental Data in a Hudi Foreign Table

You can set incremental query parameters to query incremental data in a Hudi foreign table.

```
SET hoodie.SCHEMA.FOREIGN_TABLE.consume.mode=incremental;  
SET hoodie.SCHEMA.FOREIGN_TABLE.consume.start.timestamp=start_timestamp;  
SET hoodie.SCHEMA.FOREIGN_TABLE.consume.ending.timestamp=end_timestamp;  
SELECT * FROM SCHEMA.FOREIGN_TABLE;
```

Example:

Query the incremental data of the MOR hudi foreign table **public.rtd_mfdt_int_currency_ft** from **20221207164617** to **20221207170234**.

```
SET hoodie.public.rtd_mfdt_int_currency_ft.consume.mode=incremental;  
SET hoodie.public.rtd_mfdt_int_currency_ft.consume.start.timestamp=20221207164617;  
SET hoodie.public.rtd_mfdt_int_currency_ft.consume.ending.timestamp=20221207170234;  
SELECT * FROM public.rtd_mfdt_int_currency_ft where _hoodie_commit_time>20221207164617 and  
_hoodie_commit_time<=20221207170234;
```

Querying the Configured Incremental Parameters in a Hudi Foreign Table

You can use the **pg_show_custom_settings()** function to view the configured incremental parameters in a Hudi foreign table and check whether the parameters are correctly set.

```
SELECT * FROM pg_show_custom_settings();
```

Querying the Properties of a Hudi Foreign Table (hoodie.properties)

Query the hoodie.properties properties of a Hudi foreign table on OBS.

Syntax:

```
SELECT * FROM hudi_get_options('SCHEMA.FOREIGN_TABLE');
```

Example: Query the hudi properties of the OBS foreign table **rtd_mfdt_int_currency_ft** in the current schema.

```
SELECT * FROM hudi_get_options('rtd_mfdt_int_currency_ft');
```

Querying the Maximum Timeline of a Hudi Foreign Table

Query the maximum timeline of Hudi data on OBS, that is, the latest commit record.

Syntax:

```
SELECT * FROM hudi_get_max_commit('SCHEMA.FOREIGN_TABLE');
```

Example: Query the maximum timeline of the OBS foreign table **rtd_mfdt_int_currency_ft** in the current schema.

```
SELECT * FROM hudi_get_max_commit('rtd_mfdt_int_currency_ft');
```

7.6.7 Accessing Hudi Tables on MRS

SQL on Hudi supports access to hudi tables stored on MRS.

Prerequisites

You have created an MRS data source. For details, see "MRS Data Sources" in *Data Warehouse Service (DWS) User Guide*.

NOTE

SQL on Hudi can read hudi tables stored on MRS. The only difference in usage compared to OBS is when creating data sources.

Accessing Multiple MRS Clusters Concurrently

Due to JDK restrictions, one JVM can store only one Kerberos configuration file at a time. As a result, one DWS cluster cannot concurrently access Hudi tables in multiple MRS clusters through SQL on Hudi. However, you can perform the following operations to access Hudi tables in multiple MRS clusters.

- Step 1** Obtain the `krb5.conf` file of each MRS cluster from the downloaded client.
- Step 2** Use the `krb5.conf` file of any MRS cluster as the file to be combined (cluster A for short).
- Step 3** Add the KDC domain information of cluster B to **realms** in cluster A's configuration file.

CAUTION

In practice, you need to combine the actual KDC domain information in **realms** or **domain_realm** of the cluster.

Example:

```
[realms]
CLUSTER.A.COM = {
  admin_server = ClusterA_SERVER_IP:PORT
  kdc = ClusterA_KDC_IP:PORT
  kdc = ClusterA_KDC_IP:PORT
}
CLUSTER.B.COM = {
  admin_server = ClusterB_SERVER_IP:PORT
  kdc = ClusterB_KDC_IP:PORT
  kdc = ClusterB_KDC_IP:PORT
}
```

- Step 4** Add the domain information of cluster B to **domain_realm** in cluster A's configuration file.

Example:

```
[domain_realm]
.cluster.a.com = CLUSTER.A.COM
.cluster.b.com = CLUSTER.B.COM
```

- Step 5** Replace the original **krb5.conf** file with the combined one in the original path of each node in each cluster.

----End

8 DWS Sorting Rules

The collation feature allows you to specify the data sorting order and data classification rules in a character set. This alleviates the restriction that the **LC_COLLATE** and **LC_CTYPE** settings of a database cannot be changed after its creation.

Overview

Every expression of a collatable data type has a collation. (The built-in collatable data types are text, varchar, and char. User-defined base types can also be marked collatable, and of course a domain over a collatable data type is collatable.) If the expression is a column reference, the collation of the expression is the defined collation of the column. If the expression is a constant, the collation is the default collation of the data type of the constant. The collation of a more complex expression is derived from the collations of its inputs.

Collation Combination Principles

- The collation of an expression can be the default collation, which means the locale settings defined for the database. It is also possible for an expression's collation to be indeterminate. In such cases, ordering operations and other operations that need to know the collation will fail.
- For a function or operator call, the collation that is derived by examining the argument collations is used at run time for performing the specified operation. If the result of the function or operator call is of a collatable data type, the collation is also used as the defined collation of the function or operator expression, in case there is a surrounding expression that requires knowledge of its collation.
- The collation derivation of an expression can be implicit or explicit. This distinction affects how collations are combined when multiple different collations appear in an expression. An explicit collation derivation occurs when a **COLLATE** clause is used; all other collation derivations are implicit. When multiple collations need to be combined, the following rules are used:
 - If any input expression has an explicit collation derivation, then all explicitly derived collations among the input expressions must be the same, otherwise an error is raised. If any explicitly derived collation is present, that is the result of the collation combination.

- Otherwise, all input expressions must have the same implicit collation derivation or the default collation. If any non-default collation is present, that is the result of the collation combination. Otherwise, the result is the default collation.
- If there are conflicting non-default implicit collations among the input expressions, then the combination is deemed to have indeterminate collation. This is not an error condition unless the particular function being invoked requires knowledge of the collation it should apply. If it does, an error will be raised at run-time.
- In a CASE expression, the comparison rule is subject to the COLLATE setting in the WHEN clause.
- Explicit COLLATE derivation takes effect only in the current query (CTE or SUBQUERY). Outside the query, implicit derivation takes effect.

Collation Tips

- Do not use multiple collations in the same query statement. Otherwise, exceptional result sets may be generated.
- Do not use multiple COLLATE clauses to specify a collation.

Case-insensitive Collation Support

In cluster 8.1.3 and later versions, the built-in case_insensitive collation is available for DWS, that is, case-insensitive behavior (such as sorting, comparison, and hashing) of character types is supported.

Constraints:

- Supported character types: char, character, nchar, and varchar/character varying/varchar2/nvarchar2/clob/text.
- The character types **char** and **name** are not supported.
- The following encoding formats are not supported: PG_EUC_JIS_2004, PG_MULE_INTERNAL, PG_LATIN10 and PG_WIN874.
- It cannot be specified to **LC_COLLATE** when **CREATE DATABASE** is executed.
- Regular expressions are not supported.
- Record comparison of the character type (for example, **record_eq**) is not supported.
- Time series tables are not supported.
- Skew optimization is not supported.
- RoughCheck optimization is not supported.

Examples

The COLLATE clause is specified in the statement.

```
SELECT 'a' = 'A', 'a' = 'A' COLLATE case_insensitive;  
?column? | ?column?  
-----+-----  
f        | t  
(1 row)
```

Set the column attribute to **case_insensitive** when creating a table.

```
CREATE TABLE t1 (a text collate case_insensitive);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
\d t1
      Table "public.t1"
Column | Type |      Modifiers
-----+-----+-----
a      | text | collate case_insensitive

INSERT INTO t1 values('a'),('A'),('b'),('B');
INSERT 0 4
```

This parameter is specified during table creation and does not need to be specified during query.

```
SELECT a, a='a' FROM t1;
a | ?column?
---+-----
A | t
B | f
a | t
b | f
(4 rows)
SELECT a, count(1) FROM t1 GROUP BY a;
a | count
---+-----
a | 2
B | 2
(2 rows)
```

CASE expression, which is subject to the COLLATE setting in the WHEN clause.

```
SELECT a, case a when 'a' collate case_insensitive then 'case1' when 'b' collate "C" then 'case2' else 'case3'
end FROM t1;
a | case
---+-----
A | case1
B | case3
a | case1
b | case2
(4 rows)
```

Implicit derivation across subqueries.

```
SELECT * FROM (SELECT a collate "C" from t1) WHERE a in ('a','b');
a
---
a
b
(2 rows)
SELECT * FROM t1, (SELECT a collate "C" from t1) t2 WHERE t1.a=t2.a;
ERROR: could not determine which collation to use for string hashing
HINT: Use the COLLATE clause to set the collation explicitly.
```




CAUTION

- **collate case_insensitive** is an insensitive sorting, and the result set is uncertain. If sensitive sorting is used after **collate case_insensitive** sorting, the result set may be unstable. Therefore, do not use sensitive sorting and insensitive sorting together in statements.
 - If **collate case_insensitive** is used to specify character behaviors as case-insensitive, the performance will be affected. If you require high performance, exercise caution when configuring this parameter.
-

9 DWS Materialized Views

9.1 Overview

A materialized view is a special database object. Materialized views calculate complex query results in advance and store the results to storage media. During service query, the pre-calculated data is directly queried to accelerate the query.

Materialized views of DWS is supported only by clusters of version 8.2.1.220 or later. To use materialized views, ensure that [enable_matview](#) is set to **on**.

Basic Concepts

Table 9-1 Basic concepts of materialized views

Concept	Description
Base table	- Tables on which the materialized view definition depends. The data of the materialized view comes from the calculation results of these tables. - Base tables can be DWS internal tables, foreign tables, partitioned tables, views, or materialized views. - Temporary tables are not supported.
Refresh	After a materialized view is created, the data stored in the materialized view reflects only the status at the creation time. When the data in the base table changes, you need to refresh the materialized view to update the data. Refresh modes include full refresh and partition refresh. - Full refresh: The data in the materialized view is recalculated each time the data is refreshed. - Partition refresh: The data of a partition in the materialized view is refreshed each time.

Concept	Description
Query rewriting	If you use materialized views to accelerate queries, when you enter a query that does not use materialized views, the system automatically rewrites the query to a query that uses materialized views. This method works by comparing the query with the materialized view definition. If a matching materialized view is identified, the query is automatically modified to use that materialized view. This allows for leveraging pre-calculated results from the materialized view, leading to significant enhancements in query performance and efficiency.

Asynchronous and Synchronous Materialized Views

When the data in the base table changes after the materialized view is created, you need to refresh the materialized view to update the data. Asynchronous materialized views and synchronous materialized views use different refresh modes.

Currently, DWS supports only asynchronous materialized views.

- Asynchronous materialized view: After the data in the base table changes, the materialized view needs to be refreshed periodically to update the data in it.
 - Manual refresh: After service data is modified, run the **REFRESH MATERIALIZED VIEW** command to refresh the corresponding materialized view.
 - Automatic refresh: Specify the refresh start time and refresh interval when the materialized view is created. The database periodically updates the data.
- Synchronous materialized view: The materialized view data automatically changes after the base table data changes.

Advantages of Materialized Views

- Higher query performance**
Materialized views pre-calculate and store query results. When a query is initiated, materialized views can be queried and directly return results, improving query performance.
- Lower compute resource consumption**
Executing complex queries multiple times requires significant CPU and memory resources. Materialized views help reduce the load of database resources by avoiding these repeated calculations.
- Data warehouse modeling**
Materialized views can be used as the data access layer to support complex service logic and allow users to access preprocessed data without considering the complex SQL logic at the bottom layer.

Materialized View Syntax

Create a materialized view.

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] materialized_view_name
[ ( column_name [, ...] ) ]
[ BUILD { DEFERRED | IMMEDIATE } ]
[ REFRESH [ [DEFAULT] | [ RESTRICT ] | [ CASCADE FORWARD ] | [ CASCADE BACKWARD ] | [ CASCADE
ALL ] ] [ COMPLETE ] [ ON DEMAND ] [ [ START WITH (timestampz) ] | [ EVERY (interval) ] ] ]
[ { ENABLE | DISABLE } QUERY REWRITE ]
[ WITH ( {storage_parameter = value} [, ...] ) ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...]) } } ]
[ PARTITION BY {(column_name) | date_trunc('time_unit', column_name)} ]
AS query;
```

Refresh a materialized view.

```
REFRESH MATERIALIZED VIEW
{[schema.]materialized_view_name} [ [DEFAULT] | [ RESTRICT ] | [ CASCADE FORWARD ] | [ CASCADE
BACKWARD ] | [ CASCADE ALL ] ]
```

Delete a materialized view.

```
DROP MATERIALIZED VIEW [ IF EXISTS ]
{[schema.]materialized_view_name} [, ...] [ CASCADE | RESTRICT ];
```

Modify a materialized view.

```
ALTER MATERIALIZED VIEW [ IF EXISTS ] { materialized_view_name }
[ ENABLE | DISABLE ] QUERY REWRITE;
ALTER MATERIALIZED VIEW [ IF EXISTS ] { materialized_view_name }
REFRESH [ [DEFAULT] | [ RESTRICT ] | [ CASCADE FORWARD ] | [ CASCADE BACKWARD ] | [ CASCADE
ALL ] ] [ COMPLETE ] [ ON DEMAND ] [ [ START WITH (timestampz) ] | [ EVERY (interval) ] ];
ALTER MATERIALIZED VIEW [ IF EXISTS ] { materialized_view_name }
REFRESH [DEFAULT] | [ RESTRICT ] | [ CASCADE FORWARD ] | [ CASCADE BACKWARD ] | [ CASCADE
ALL ] ;
ALTER MATERIALIZED VIEW { materialized_view_name }
OWNER TO new_owner;
ALTER MATERIALIZED VIEW { materialized_view_name }
SET ( {storage_parameter = value} [, ...] )
| RESET ( storage_parameter [, ...] )
```

For details about the parameters of the materialized view syntax, see "CREATE MATERIALIZED VIEW" and "ALTER MATERIALIZED VIEW" in *SQL Syntax Reference*.

Application Scenarios

Materialized views apply to the following scenarios:

- The query result does not change frequently, that is, the base table is stable and the execution frequency is high.
- The query involves time-consuming operations, such as aggregation and join. For queries involving multiple table join and aggregation operations, materialized views can pre-calculate and store these time-consuming operations to avoid re-executing these operations during each query.
- The query result involves only a small part of data in the table.

Table 9-2 Comparison between materialized views, views, and tables

Database Object	Performance	Persistent Storage	Simplified Service Logic
Materialized views	√	√	√
Views	-	-	√
Tables	-	√	-

Permissions

- To create a materialized view, you must have the **CREATE** permission on the schema and the **SELECT** permission on the base table or column.
- To query a materialized view, you must have the **SELECT** permission on the materialized view.
- To refresh a materialized view, you must have the **INSERT** permission on the view and the **SELECT** permission on the base table or column.
- To delete a materialized view, you must have the **DROP TABLE** permission.

Constraints

- Temporary tables cannot be included, including global, volatile, and common temporary tables.
- The CTE in the definition of a materialized view cannot contain other DML statements except the **SELECT** statement.
- The base table of a materialized view cannot have data masking policies or row-level access control. It also cannot be owned by a private user. The usage of masking policies and row-level access control may lead to errors in the result set, while materialized views owned by private users may cause refresh failures.
- 9.1.0.200 and later versions support the **stable** and **volatile** functions. Note that materialized views cannot detect function content changes. If function changes need to be fed back to the materialized view as soon as possible, specify **REFRESH MATERIALIZED VIEW RESTRICT** to refresh the materialized view or feed back the data changes to the materialized view.
- The materialized view cannot detect the parameter modification that affects the query execution. Therefore, you need to refresh the materialized view again.

Usage Suggestions

- Storage resources: Materialized views occupy extra storage space. Therefore, you need to consider the disk capacity.
- Refresh maintenance: Refreshing the materialized view, whether done automatically or manually, uses system resources. If the data in the base table changes often, the refresh process can impact system performance.
- Asynchronous materialized views do not update automatically when the original data changes. If the base table data is modified, the view data may

become outdated. You need to periodically update the view data to ensure data consistency.

- Design complexity: When designing and creating materialized views, it is crucial to consider the expected query mode and data access mode to optimize performance.

9.2 Asynchronous Materialized Views

9.2.1 Materialized View Query Rewriting

9.2.1.1 Overview

DWS supports query of materialized views and query rewriting. Materialized view query rewriting allows users to speed up query execution using materialized views automatically, without altering services. This feature lets you add or remove materialized views similar to indexes, without impacting services.

Scenarios

Materialized view query rewriting can avoid complex association, aggregation, and expression calculation operations in the query.

Text matching query rewriting allows queries to match the definition of a materialized view, even if the format and comments differ. It supports complex queries like CTE and window functions. If a complex query performs poorly, creating a materialized view with the same statement can speed up the query.

Query Rewriting Methods

Text matching query rewriting

When the query matches the SQL or syntax tree of the materialized view, you can use the materialized view to rewrite the query.

Setting Materialized View Query Rewriting

The GUC parameter [mv_rewrite_rule](#) of DWS specifies the query rewriting mode.

9.2.1.2 Text Matching Query Rewriting

DWS supports text matching query rewriting, including complex queries such as **JOIN**, **AGG**, **CTE**, and subqueries.

Constraints

- Subqueries of a query can also use materialized views for text matching.
- If the query statement contains **ORDER BY**, the **ORDER BY** column must exist in the output column of the materialized view.
- The distinctions between a query and a materialized view can include spaces, line breaks, and comments.

- If an alias is assigned to the output column of a materialized view, the same alias must be used in the query for proper matching.
- Queries in materialized views cannot contain **FOR UPDATE**.

Examples

Take tpch Q1 as an example. When a materialized view is generated from the intricate query in Q1, the query plan indicates that the statement has been modified to access the materialized view.

```
CREATE MATERIALIZED VIEW mv_tpch_q1 enable query rewrite as
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) AS sum_qty,
    sum(l_extendedprice) AS sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    avg(l_quantity) AS avg_qty,
    avg(l_extendedprice) AS avg_price,
    avg(l_discount) AS avg_disc,
    count(*) AS count_order
FROM
    lineitem
WHERE l_shipdate <= DATE '1998-12-01' - INTERVAL '3 day'
GROUP BY l_returnflag, l_linestatus
ORDER BY sum_qty;

explain (costs off)
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) AS sum_qty,
    sum(l_extendedprice) AS sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    avg(l_quantity) AS avg_qty,
    avg(l_extendedprice) AS avg_price,
    avg(l_discount) AS avg_disc,
    count(*) AS count_order
FROM
    lineitem
WHERE l_shipdate <= DATE '1998-12-01' - INTERVAL '3 day'
GROUP BY l_returnflag, l_linestatus
ORDER BY sum_qty;
      QUERY PLAN
-----
id |          operation
---+-----
 1 | -> Streaming (type: GATHER)
 2 | -> Sort
 3 | -> Seq Scan on mv_tpch_q1
```

If the **ORDER BY** section aligns with the materialized view definition, complete matching can also be executed.

```
CREATE TABLE t1 (a int, b int);
CREATE MATERIALIZED VIEW mv_t1 enable query rewrite AS SELECT sum(a), b FROM t1 GROUP BY b;

EXPLAIN (costs off) SELECT sum(a), b FROM t1 GROUP BY b ORDER BY 1;
      QUERY PLAN
-----
id |          operation
---+-----
 1 | -> Streaming (type: GATHER)
 2 | -> Sort
 3 | -> Seq Scan on mv_t1
```

9.2.2 Partitioned Materialized Views

9.2.2.1 Overview

To quickly update materialized views by partition, you can create partition tables for the views and establish mappings between the views and the base tables. In this way, when certain partitions of the base table change, the refresh process can target only the affected partitions, bringing in the latest data to the corresponding partitions of the materialized view.

- **View creation**

Designating the partition key of the queried base table as the partition key of the materialized view can implement the automatic partition mapping between the materialized view and the base table.

- **Partition invalidation**

- If data in a partition of the base table changes, the corresponding partition in the materialized view is automatically marked as invalid.
- Similarly, when data in the base table is modified concurrently, the materialized view flags are invalidated concurrently to prevent repeated marking.

- **Incremental update**

- During a refresh of a materialized view, only invalid partitions are automatically identified and refreshed.
- If a partition is currently being refreshed concurrently, the system will skip the refresh to avoid repeated refresh.
- Updating materialized views does not interfere with data modifications in the base table.

- **Partition DDL**

When a partition is added or deleted in the base table, the corresponding partition in the materialized view is also added or deleted according to the established mapping rule.

9.2.2.2 Creating a Partitioned Materialized View

DWS allows you to create partitioned materialized views in the following scenarios:

- Partitions are aligned proportionally. The partitioning conditions of materialized views are the same as those of base tables.
- Partitions are aligned based on time granularity, for example, daily partitioning for the base table and monthly partitioning for the materialized view.
- Partitions are aligned for multiple base tables with identical partition conditions, automatically mapping the base tables to the materialized view.
- The base table has multiple partition keys and the first key is chosen as the partition key for creating a materialized view.

Syntax

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] materialized_view_name
[ ( column_name [, ...] ) ]
[ BUILD { DEFERRED | IMMEDIATE } ]
[ REFRESH [ [DEFAULT] | [ RESTRICT ] | [ CASCADE FORWARD ] | [ CASCADE BACKWARD ] | [ CASCADE
ALL ] ] [ COMPLETE ] [ ON DEMAND ] [ [ START WITH (timestamp) ] | [ EVERY (interval) ] ] ]
[ { ENABLE | DISABLE } QUERY REWRITE ]
[ WITH ( {storage_parameter = value} [, ...] ) ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...] ) } } ]
[ PARTITION BY {(column_name) | date_trunc('time_unit', column_name)} ]
AS query;
```

Constraints

- You can only use range partitioning.
- The materialized view only supports a single partition key, which must be the first partition key of the base table. The base table can have a combined partition key.
- Rollup mode is supported only for time type, using **date_trunc** for rollup calculation. Supported rollup intervals include day, month, quarter, and year.
- Columns that generate null values in outer joins cannot be used as partition keys.
- **disable full group by** in MySQL compatibility mode is not supported.
- Pre-aggregated materialized views cannot be used to create partitioned materialized views.
- Query rewriting for a specified partition is not allowed.
- The base table of the partitioned materialized view does not support **SPLIT** and **MERGE** operations.
- When both sides of a union meet the partition constraint check of the materialized view, a materialized view bound to multiple tables is created.
- If **JOIN** is used to join base tables and the partition key is set to a public column, a materialized view bound to multiple tables is created if the joined base tables meet the partition constraint detection of the materialized view.
- Query statements cannot contain sublinks.
- Query statements cannot contain subplans.
- Data modification using DML and DDL statements is not allowed. Data can only be updated by refreshing.

Examples

Create fact tables **fact_table** and **fact_table1**.

```
CREATE TABLE fact_table (start_time TIMESTAMPTZ NOT NULL, id int)PARTITION BY RANGE(start_time)(
PARTITION p1 VALUES LESS THAN ('2024-01-01'),
PARTITION p2 VALUES LESS THAN ('2024-01-02'),
PARTITION p3 VALUES LESS THAN ('2024-02-03'),
PARTITION p4 VALUES LESS THAN ('2024-03-01'),
PARTITION p5 VALUES LESS THAN ('2024-03-21'),
PARTITION p6 VALUES LESS THAN ('2024-04-02'));
CREATE TABLE fact_table1 (start_time TIMESTAMPTZ NOT NULL, id int)PARTITION BY RANGE(start_time)(
PARTITION p1 VALUES LESS THAN ('2024-01-01'),
PARTITION p2 VALUES LESS THAN ('2024-01-02'),
PARTITION p3 VALUES LESS THAN ('2024-02-03'),
PARTITION p4 VALUES LESS THAN ('2024-03-01'),
```

```
PARTITION p5 VALUES LESS THAN ('2024-03-21'),  
PARTITION p6 VALUES LESS THAN ('2024-04-02'));
```

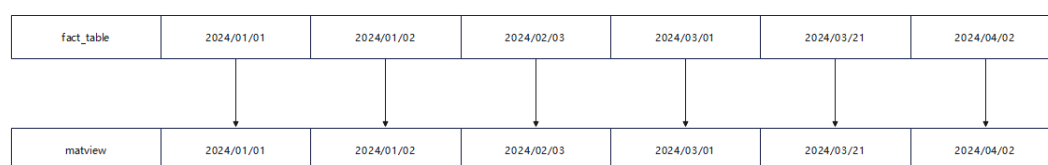
Create a dimension table **dimen_table**.

```
CREATE TABLE dimen_table (start_time TIMESTAMPTZ NOT NULL, id INT);
```

Scenario 1: Align partitions proportionally.

Create the materialized view **level_normal**. Ensure that the partition of the materialized view matches that of the fact_table table, with identical partition conditions and no specific data type requirements.

```
CREATE MATERIALIZED VIEW level_normal enable query rewrite  
DISTRIBUTE BY HASH(start_time)  
PARTITION BY start_time  
AS SELECT fact_table.start_time, dimen_table.id FROM fact_table JOIN dimen_table ON fact_table.id =  
dimen_table.id;
```

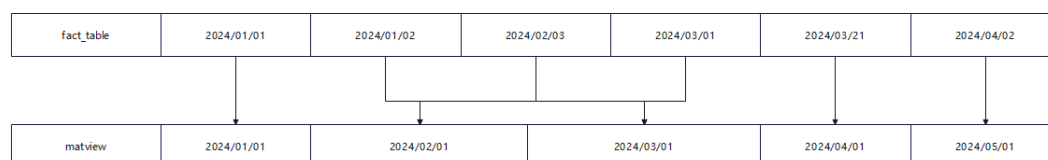


Scenario 2: Adjust the partitioning time granularity.

Use the **date_trunc** parameter on the partition key to create a materialized view with coarser partitioning time granularity compared to the base table. DWS updates the corresponding roll-up partition in the materialized view after detecting data changes in the base table partition.

You can use the **date_trunc('time_unit', column_name)** parameter to specify the roll-up targets. The materialized view partition covers a wider time range compared to the base table partition. For instance, if the base table is partitioned by day, the materialized view can be partitioned by month. The **date_trunc** parameter adjusts the time granularity in the order of day, quarter, year, or month. Supported data types include time, timestamp, and timestamptz.

```
CREATE MATERIALIZED VIEW level_month enable query rewrite  
DISTRIBUTE BY HASH(start_time)  
PARTITION BY date_trunc('month', start_time)  
AS SELECT fact_table.start_time, dimen_table.id FROM fact_table JOIN dimen_table ON fact_table.id =  
dimen_table.id;
```

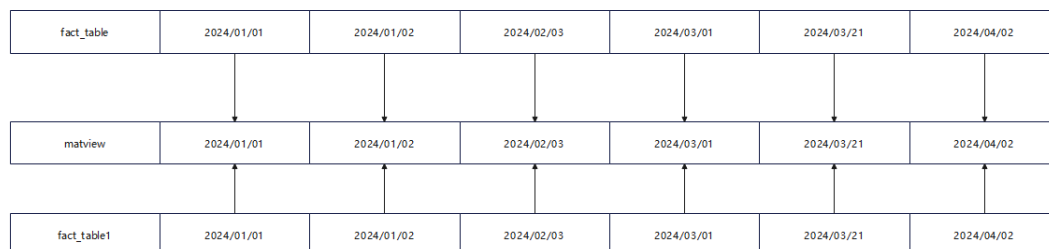


Scenario 3: Align partitions for multiple base tables.

If multiple base tables use the same partition key type and partition range, you can create a materialized view bound to multiple base tables.

```
CREATE MATERIALIZED VIEW level_normal enable query rewrite  
DISTRIBUTE BY HASH(start_time)  
PARTITION BY start_time  
AS SELECT start_time, id FROM fact_table union SELECT start_time, id FROM fact_table1;  
CREATE MATERIALIZED VIEW level_normal enable query rewrite  
DISTRIBUTE BY HASH(start_time)  
PARTITION BY start_time
```

```
AS SELECT fact_table1.start_time, fact_table.id, fact_table1.id as id1
FROM fact_table inner join fact_table1 on fact_table.start_time=fact_table1.start_time;
```



Scenario 4: When the base table has multiple partition keys, use the first partition key of the base table to create a materialized view.

```
CREATE TABLE IF NOT EXISTS multi_time_base (
  new_start_time timestamp with time zone NOT NULL,
  id int,
  v1 int
) PARTITION BY RANGE(new_start_time,id) (
  PARTITION p1 VALUES LESS THAN ('2022-01-01',400),
  PARTITION p2 VALUES LESS THAN ('2022-01-05',400),
  PARTITION p3 VALUES LESS THAN ('2022-01-05',500),
  PARTITION p4 VALUES LESS THAN ('2022-02-15',500),
  PARTITION p5 VALUES LESS THAN ('2022-02-16',500),
  PARTITION p6 VALUES LESS THAN ('2022-03-10',500)
);
CREATE MATERIALIZED VIEW multi_time_mv
PARTITION BY (date_trunc('month', new_start_time))
AS SELECT new_start_time,id FROM multi_time_base;
```

9.2.2.3 Refreshing a Partition Materialized View

Using partition refresh reduces the data update costs after a partitioned materialized view is created.

Enabling Auto Refresh

The system automatically identifies and refreshes the materialized view. You can run the following command to trigger a refresh:

```
REFRESH MATERIALIZED VIEW materialized_view_name;
```

This command initiates a refresh of the materialized view. If the conditions for partition refresh are met, the system refreshes the partition; otherwise, a full refresh is performed.

Manually Refreshing a Specified Partition

Run the following command to specify the materialized view partition to be refreshed:

```
REFRESH MATERIALIZED VIEW materialized_view_name PARTITION (partname1,partname2);
```

This command refreshes the specified partitions, **partname1** and **partname2**, of the materialized view. If the current partition does not require refreshing, the system displays a message indicating that the data is up to date.

Forcibly and Manually Refreshing a Specified Partition

Run the following command to forcibly refresh the specified partition of the materialized view:

```
REFRESH MATERIALIZED VIEW materialized_view_name PARTITION (partname) RESTRICT;
```

Viewing a Partitioned Materialized View

Check whether the partition data is updated.

Use the **active** column of **pg_matview_partition** to check whether the current partition has been updated.

```
\d pg_matview_partition;
Table "pg_catalog.pg_matview_partition"
  Column      |      Type      | Modifiers
-----+-----+-----
 matpartid    |      oid       | not null
  active      |      boolean   | not null
 refresh_start_time | timestamp with time zone | not null
 refresh_finish_time | timestamp with time zone | not null
```

Where,

- **matpartid** indicates the OID of the materialized view partition.
- **active** indicates whether the current partition has been updated. **t** indicates updated and **f** indicates invalid.
- **refresh_start_time** indicates the start time of the last partition refresh.
- **refresh_finish_time** indicates the end time of the last partition refresh.

Examples

Step 1 Create the materialized view **level_month** and check the view information.

```
CREATE MATERIALIZED VIEW level_month enable query rewrite
DISTRIBUTE BY HASH(start_time)
PARTITION BY date_trunc('month', start_time)
AS SELECT fact_table.start_time, dimen_table.id FROM fact_table JOIN dimen_table ON fact_table.id =
dimen_table.id;
SELECT * FROM pg_matview_partition;
 matpartid | active | refresh_start_time | refresh_finish_time
-----+-----+-----+-----
 17092 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17093 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17094 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17095 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17096 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
```

Step 2 Insert data into the base table.

```
INSERT INTO fact_table (start_time, id) VALUES ('2023-05-01',1);
```

Step 3 Check the partition invalidation information.

```
SELECT * FROM pg_matview_partition;
 matpartid | active | refresh_start_time | refresh_finish_time
-----+-----+-----+-----
 17092 | f | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17093 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17094 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17095 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
 17096 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08
```

Step 4 Refresh a materialized view.

```
REFRESH MATERIALIZED VIEW level_month;
```

Partition refresh is triggered because the problem is partition invalidity.

Step 5 Check the status after the refresh.

```
SELECT * FROM pg_matview_partition;  
matpartid | active | refresh_start_time | refresh_finish_time  
-----+-----+-----+-----  
17093 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08  
17094 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08  
17095 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08  
17096 | t | 2024-11-18 15:15:32.177697+08 | 2024-11-18 15:15:32.204009+08  
17092 | t | 2024-11-18 15:16:28.902902+08 | 2024-11-18 15:16:28.940159+08
```

----End

9.2.3 Nested Materialized Views

9.2.3.1 Overview

DWS materialized views support nested creation, refreshing, and rewriting. Using nested materialized views can simplify the process of defining a single view.

- Creating a nested materialized view: You can build a materialized view based on the materialized view.
- Cascading refresh: When refreshing a nested materialized view, not only the current view is refreshed, but automatic cascading refresh is also supported.
- Scheduled cascading refresh: You can specify and modify the refresh mode for periodically refreshed materialized views.
- Rewriting nested materialized views: Query statements can be automatically rewritten as SQL statements for querying materialized views.

9.2.3.2 Creating a Nested Materialized View

DWS enables the creation of nested materialized views, which involve using other materialized views in defining a view. By nesting materialized views, you can merge multiple views to generate more intricate and comprehensive query results.

The GUC parameter [mv_max_cascade_depth](#) is used to specify the maximum depth of a nested materialized view. To ensure refresh performance, the maximum depth is 10. The default value is 3.

Constraints

1. Nesting of pre-aggregated views is not allowed.
2. Pre-aggregated materialized views cannot be based on other materialized views.
3. The refresh mode of a materialized view cannot be set or changed unless it is refreshed periodically.
4. The maximum depth for nested materialized views is 10, with no support for deeper nesting layers currently.

5. Nested materialized views cannot be created during an upgrade.

Syntax

The syntax for creating a nested materialized view is similar to creating a regular materialized view. In the view definition statement, the base table can be another materialized view. Use **refresh_behavior** to schedule periodic refreshes and specify the refresh mode.

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] materialized_view_name
[ ( column_name [, ...] ) ]
[ BUILD { DEFERRED | IMMEDIATE } ]
[ REFRESH [ refresh_behavior ] [ COMPLETE ] [ ON DEMAND ] [ [ START WITH (timestampz) ] | [ EVERY
(interval) ] ] ]
[ { ENABLE | DISABLE } QUERY REWRITE ]
[ WITH ( {storage_parameter = value} [, ...] ) ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...] ) } } ]
[ PARTITION BY {(column_name) | date_trunc('time_unit', column_name)} ]
AS query;
```

Viewing a Nested Materialized View

The **relid** column in the **pg_matview_depend** system catalog stores the OIDs of materialized views.

```
\d+ pg_matview_depend
Table "pg_catalog.pg_matview_depend"
Column          | Type          | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
matviewwid      | oid           | not null  | plain   |              |
relid           | oid           | not null  | plain   |              |
partids         | oidvector     | not null  | plain   |              |
contain_entire_rel | boolean       | not null  | plain   |              |
Indexes:
"pg_matview_depend_mvid_relid_index" UNIQUE, btree (matviewwid, relid) TABLESPACE pg_default
"pg_matview_depend_relid_index" btree (relid) TABLESPACE pg_default
Replica Identity: NOTHING
Has OIDs: no
```

Examples

1. Create base tables, materialized views, and nested materialized views.

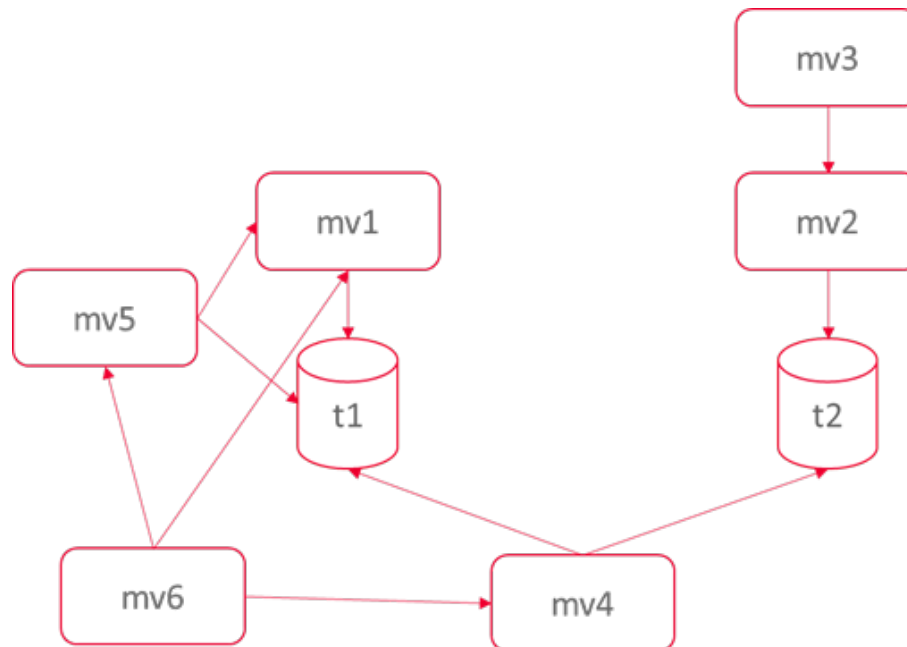
```
CREATE TABLE t1(a int, b int, c int);
CREATE TABLE t2(a int, b int, c int);
CREATE MATERIALIZED VIEW mv1 ENABLE QUERY REWRITE as select * from t1 WHERE a = 1;
CREATE MATERIALIZED VIEW mv2 ENABLE QUERY REWRITE as select * from t2 WHERE b = 1;
CREATE MATERIALIZED VIEW mv3 ENABLE QUERY REWRITE as select * from mv2 WHERE b = 1;
CREATE MATERIALIZED VIEW mv4 ENABLE QUERY REWRITE as select t1.a as t1a,t1.b as t1b,t1.c as t1c,t2.a
as t2a,t2.b as t2b,t2.c as t2c from t1 left join t2 on t1.a = t2.a WHERE t2.b = 1;
CREATE MATERIALIZED VIEW mv5 ENABLE QUERY REWRITE as (select * from mv1) union all (select * from
t1 WHERE a = 1);
CREATE MATERIALIZED VIEW mv6 ENABLE QUERY REWRITE as select * from mv4 left join (select * from
mv1 union all select * from mv5)mv5 on mv4.t1a = mv5.a;
```

2. In the SQL statement above, tables **t1** and **t2** are created, followed by the creation of materialized views **mv1** and **mv2** based on **t1** and **t2**. Subsequently, create materialized views **mv3**, **mv4**, **mv5**, and **mv6** sequentially using **t1**, **t2**, **mv1**, and **mv2**.

Nested materialized views offer a simpler definition and more flexible usage compared to single-layer materialized views.

3. The figure below illustrates the creation relationship and dependency between base tables and materialized views.

Figure 9-1 Dependency between nested materialized views



4. When there are changes in the data at the lowest level, you must refresh the materialized view to ensure its data validity. For example, changes in the **t1** table require updates in the **mv1**, **mv4**, **mv5**, and **mv6** tables. Refreshing a single materialized view in cascading mode automatically updates all associated materialized views.

```
REFRESH MATERIALIZED VIEW mv6 cascade backward;
```

5. Once all data in a nested materialized view is valid, the system automatically matches and retrieves the data when executing a query statement. It then automatically converts the query statement into a materialized view query, significantly reducing the computational load in the query statement.

9.2.3.3 Refreshing a Nested Materialized View

Cascading Refresh

To simplify the process of refreshing materialized views, nested materialized views can be automatically refreshed in cascading mode.

There are five supported refresh modes: default, forcible, forward cascading, backward cascading, and full cascading. This means that the **refresh_behavior** value can be set to **DEFAULT**, **RESTRICT**, **CASCADE FORWARD**, **CASCADE BACKWARD**, or **CASCADE ALL**.

- **DEFAULT**: Refresh a single materialized view. If there are no changes in the base table data, the refresh operation is skipped.
- **RESTRICT**: Force a refresh on a single materialized view.

- **CASCADE FORWARD:** Refresh the current materialized view, followed by refreshing the nested materialized views that depend on it in sequence.
- **CASCADE BACKWARD:** Identify all materialized views that the current view depends on, refresh them from the bottom layer, and then refresh the current view.
- **CASCADE ALL:** Perform backward cascading refresh and then forward cascading refresh.

 NOTE

If you do not specify a cascading refresh mode, the system will use the default mode, which skips the refresh operation when there are no changes in the data of lower-layer materialized views or tables.

Example:

The nested materialized view created in [Creating a Nested Materialized View](#) is used as an example.

Perform a cascading refresh on the materialized view **mv5**.

```
REFRESH MATERIALIZED VIEW mv5 cascade all;
```

The cascading refresh process involves refreshing **mv1**, **mv5**, and **mv6** sequentially.

This is achieved by performing a backward cascading refresh on **mv5** followed by a forward cascading refresh.

```
REFRESH MATERIALIZED VIEW mv5 cascade backward;  
REFRESH MATERIALIZED VIEW mv5 cascade forward;
```

In a backward cascading refresh, all bottom-layer materialized views that **mv5** depends on are identified first, and then the materialized views are refreshed layer by layer from the bottom to the top.

```
REFRESH MATERIALIZED VIEW mv5 cascade backward;
```

Perform the cascading refresh on **mv1** first, then **mv5**.

For cascading refresh, all materialized views dependent on **mv5** are identified first, and then the materialized views are refreshed layer by layer from the bottom to the top.

Perform the cascading refresh on **mv5** first, then **mv6**.

Periodically Refreshing Materialized Views

When setting up materialized views that are refreshed periodically, you must define the refresh mode. If not specified, the default mode is used, and cascading refresh is not carried out.

Therefore, for materialized views, you can specify the cascading refresh mode during creation and make changes as needed.

Examples:

1. Create a scheduled task for refreshing materialized views.

```
CREATE TABLE test_table(a int2, b int, c int8, d int) WITH (ORIENTATION = COLUMN, COLVERSION = 2.0) DISTRIBUTE BY hash(c);  
create materialized view mv_test_sync_default REFRESH START WITH('2020-01-01 12:12:12'::timestampz) EVERY (interval '1 s') ENABLE QUERY REWRITE as
```



```
select a,sum(b) b from test_table group by a;
create materialized view mv_test_sync_restrict REFRESH restrict START WITH('2020-01-01
12:12:12::timestampz) EVERY (interval '1 s') ENABLE QUERY REWRITE as
select a,sum(b) b from test_table group by a;
create materialized view mv_test_sync_forward REFRESH cascade forward START WITH('2020-01-01
12:12:12::timestampz) EVERY (interval '1 s') ENABLE QUERY REWRITE as
select a,sum(b) b from test_table group by a;
create materialized view mv_test_sync_backward REFRESH cascade backward START
WITH('2020-01-01 12:12:12::timestampz) EVERY (interval '1 s') ENABLE QUERY REWRITE as
select a,sum(b) b from test_table group by a;
create materialized view mv_test_sync_all REFRESH cascade all START WITH('2020-01-01
12:12:12::timestampz) EVERY (interval '1 s') ENABLE QUERY REWRITE as
select a,sum(b) b from test_table group by a;
```

2. View the scheduled update task.

```
SELECT what, category_id, is_broken, interval FROM scheduler.pg_task WHERE what LIKE '%mv_test%'
ORDER BY 1;
```

```
postgres=# SELECT what, category_id, is_broken, interval FROM scheduler.pg_task WHERE what LIKE '%mv_test%' ORDER BY 1;
          what          | category_id | is_broken | interval
-----+-----+-----+-----
REFRESH MATERIALIZED VIEW public.mv_test_sync_all CASCADE ALL; | MatViewKernelTask | f | 00:00:01
REFRESH MATERIALIZED VIEW public.mv_test_sync_backward CASCADE BACKWARD; | MatViewKernelTask | f | 00:00:01
REFRESH MATERIALIZED VIEW public.mv_test_sync_default; | MatViewKernelTask | f | 00:00:01
REFRESH MATERIALIZED VIEW public.mv_test_sync_forward CASCADE FORWARD; | MatViewKernelTask | f | 00:00:01
REFRESH MATERIALIZED VIEW public.mv_test_sync_restrict RESTRICT; | MatViewKernelTask | f | 00:00:01
(5 rows)
```

3. Modify the cascading refresh mode.

```
ALTER MATERIALIZED VIEW mv_test_sync_backward REFRESH START WITH('2020-01-01
15:15:15::timestampz) EVERY (interval '10 s') ;
ALTER MATERIALIZED VIEW mv_test_sync_restrict REFRESH cascade all START WITH('2020-01-01
12:12:12::timestampz) EVERY (interval '20 s') ;
ALTER MATERIALIZED VIEW mv_test_sync_all REFRESH default;
ALTER MATERIALIZED VIEW mv_test_sync_forward REFRESH default START WITH('2020-01-01
12:12:12::timestampz) EVERY (interval '50 s') ;
```

```
postgres=# SELECT what, category_id, is_broken, interval FROM scheduler.pg_task WHERE what LIKE '%mv_test%' ORDER BY 1;
          what          | category_id | is_broken | interval
-----+-----+-----+-----
REFRESH MATERIALIZED VIEW public.mv_test_sync_all; | MatViewKernelTask | f | 00:00:01
REFRESH MATERIALIZED VIEW public.mv_test_sync_backward CASCADE BACKWARD; | MatViewKernelTask | f | 00:00:10
REFRESH MATERIALIZED VIEW public.mv_test_sync_default; | MatViewKernelTask | f | 00:00:01
REFRESH MATERIALIZED VIEW public.mv_test_sync_forward; | MatViewKernelTask | f | 00:00:50
REFRESH MATERIALIZED VIEW public.mv_test_sync_restrict CASCADE ALL; | MatViewKernelTask | f | 00:00:20
(5 rows)
```

10 DWS User-Defined Functions

10.1 DWS PL/Java Functions

DWS supports Java user-defined functions (UDFs) to extend data processing capabilities. Developers can write UDFs in Java and use them in SQL queries or other data processing tasks.

Constraints

Java UDF can be used for some Java logical computing. You are not advised to encapsulate services in Java UDF.

- You are not advised to connect to a database in any way (for example, JDBC) in Java functions.
- Currently, only data types listed in [Table 10-2](#) are supported. Other data types, such as user-defined data types and complex data types (for example, Java array and its derived types) are not supported.
- Currently, UDAF and UDTF are not supported.
- DWS PL/Java is based on open source PL/Java 1.5.5 and uses JRE 1.8.0.432.

Using Java UDFs

With PL/Java, you can compile Java methods using a Java IDE, deploy the JAR files to the DWS database, and create functions as a database administrator. For compatibility, use JRE 1.8.0.432 for compiling.

Step 1 Compile a JAR package.

Java method implementation and JAR package archiving can be achieved in an integrated development environment (IDE). The following is a simple example of compilation and archiving through command lines. You can create a JAR package that contains a single method in the similar way.

First, compile an **Example.java** file (where the **static** keyword is used). The file contains a method for converting substrings to uppercase. In the following example, **Example** is the class name and **upperString** is the method name:

```
public class Example
{
    public static String upperString (String text, int beginIndex, int endIndex)
    {
        return text.substring(beginIndex, endIndex).toUpperCase();
    }
}
```

Then, create the **manifest.txt** file. The file content is as follows. [Table 10-1](#) describes the parameters.

```
Manifest-Version: 1.0
Main-Class: Example
Specification-Title: "Example"
Specification-Version: "1.0"
Created-By: 1.6.0_35-b10-428-11M3811
Build-Date: 08/14/2018 10:09 AM
```

Table 10-1 manifest.txt file

Parameter	Description	Example
Manifest-Version	Version of the manifest file. The value is fixed at 1.0, which is compatible with the JAR specification. Retain the default value.	1.0
Main-Class	Entry class of the JAR package (class that contains the main method), that is, the class name of the main application. When the JAR package is executed using java -jar , this class is called.	Example
Specification-Title	Name identifier of the project or module, which is used to describe the function of the JAR package. In this example, the substring is converted to uppercase.	Example
Specification-Version	Semantic version of the project. The version number is user-defined. For example, the version number is 1.0 for the first release and 2.0 for subsequent iterations.	1.0
Created-By	Version and vendor of the Java that generates the manifest file. The attribute is generated by the JAR tool and is in the format of <i>JDK version number-build number</i> . - The value is automatically filled by the build tool (such as Maven) by default. Generally, you do not need to manually modify the value. - Ensure that the JDK versions in the build environment and running environment are the same.	1.6.0_35-b10-428-11M3811

Parameter	Description	Example
Build-Date	JAR package build timestamp, which is used to trace the build history for rollback or debugging.	08/14/2018 10:09 AM

Finally, archive the .java file and package it into **javaudf-example.jar**.

```
javac Example.java
jar cfm javaudf-example.jar manifest.txt Example.class
```

NOTICE

JAR package names must comply with JDK rules. If a name contains invalid characters, an error occurs when a function is deployed or used.

Step 2 Deploy the JAR package.

Place the JAR package on the OBS server using the method described in For details, see "Uploading a File" in *Object Storage Service Console Operation Guide*. Then, create the AK/SK. For details about how to obtain the AK/SK, see section [Creating Access Keys \(AK and SK\)](#). Log in to the database and run the **gs_extend_library** function as an administrator to import the file to DWS.

```
SELECT gs_extend_library('addjar', 'obs://bucket/path/javaudf-example.jar
accesskey=access_key_value_to_be_replaced secretkey=secret_access_key_value_to_be_replaced
region=region_name libraryname=example');
```

For details about how to use the **gs_extend_library** function, see [Manage JAR packages and files](#). Change the values of AK and SK as needed. Replace *region_name* with an actual region name. **bucket** indicates the name of the bucket on the OBS server.

Step 3 Use a PL/Java function.

Log in to the database as a user who has the **sysadmin** permission (for example, dbadmin) and create the **java_upperstring** function:

```
CREATE FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER)
RETURNS VARCHAR
AS 'Example.upperString'
LANGUAGE JAVA;
```

NOTE

- The data type defined in the **java_upperstring** function should be a type in DWS and match the data type defined in [Step 1](#) in the **upperString** method in Java. For details about the mapping between DWS and Java data types, see [Table 10-2](#).
- The AS clause specifies the class name and static method name of the Java method invoked by the function. The format is *Class name.Method name*. The class name and method name must match the Java class and method defined in [Step 1](#).
- To use PL/Java functions, set **LANGUAGE** to **JAVA**.
- For details about CREATE FUNCTION, see [Create functions](#).

Execute the **java_upperstring** function.

```
SELECT java_upperstring('test', 0, 1);
```

The expected result is as follows:

```
java_upperstring
-----
T
(1 row)
```

Step 4 Authorize a common user to use the PL/Java function.

Create a common user named **udf_user**.

```
CREATE USER udf_user PASSWORD 'password';
```

This command grants user **udf_user** the permission for the `java_upperstring` function. Note that the user can use this function only if it also has the permission for using the schema of the function.

```
GRANT ALL PRIVILEGES ON SCHEMA public TO udf_user;
GRANT ALL PRIVILEGES ON FUNCTION java_upperstring(VARCHAR, INTEGER, INTEGER) TO udf_user;
```

Log in to the database as user **udf_user**.

```
SET SESSION SESSION AUTHORIZATION udf_user PASSWORD 'password';
```

Execute the `java_upperstring` function.

```
SELECT public.java_upperstring('test', 0, 1);
```

The expected result is as follows:

```
java_upperstring
-----
T
(1 row)
```

Step 5 Delete the function.

If you no longer need this function, delete it.

```
DROP FUNCTION java_upperstring;
```

Step 6 Uninstall the JAR package.

Use the `gs_extend_library` function to uninstall the JAR package.

```
SELECT gs_extend_library('rmjar', 'libraryname=example');
```

----End

Mapping for Basic Data Types

Table 10-2 PL/Java mapping for default data types

DWS	Java
BOOLEAN	boolean
"char"	byte
bytea	byte[]
SMALLINT	short

DWS	Java
INTEGER	int
BIGINT	long
FLOAT4	float
FLOAT8	double
CHAR	java.lang.String
VARCHAR	java.lang.String
TEXT	java.lang.String
name	java.lang.String
DATE	java.sql.Timestamp
TIME	java.sql.Time (stored value treated as local time)
TIMETZ	java.sql.Time
TIMESTAMP	java.sql.Timestamp
TIMESTAMPTZ	java.sql.Timestamp

UDF Example: SQL Definition and Usage

- **Manage JAR packages and files.**

A database user having the **sysadmin** permission can use the `gs_extend_library` function to deploy, view, and delete JAR packages in the database. The syntax of the function is as follows:

```
SELECT gs_extend_library('[action]', '[operation]');
```

 NOTE

- action:** operation action. The options are as follows:
 - ls:** Displays JAR packages in the database and checks the MD5 value consistency of files on each node.
 - addjar:** deploys a JAR package on the OBS server in the database.
 - rmjar:** Deletes JAR packages from the database.
- operation:** operation string. The format can be either of the following:
`obs://[bucket]/[source_filepath] accesskey=[accesskey] secretkey=[secretkey] region=[region] libraryname=[libraryname]`
 - bucket:** name of the bucket to which the OBS file belongs. It is mandatory.
 - source_filepath:** file path on the OBS server. Only .jar files are supported.
 - accesskey:** key obtained for accessing the OBS service. It is mandatory.
 - secret_key:** secret key obtained for the OBS service. It is mandatory.
 - region:** region where the OBS bucket stored in the JAR package of a user-defined function belongs to. This parameter is mandatory.
 - libraryname:** user-defined library name, which is used to invoke JAR files in DWS. If **action** is set to **addjar** or **rmjar**, **libraryname** must be specified. If **action** is set to **ls**, **libraryname** is optional. Note that a user-defined library name cannot contain the following characters: `/;&$<>\'{}"() []~*?!`
- Create functions.**
 PL/Java functions can be created using the **CREATE FUNCTION** syntax and are defined as **LANGUAGE JAVA**, including the **RETURNS** and **AS** clauses.
 - To use **CREATE FUNCTION**, specify the name and parameter type for the function to be created.
 - The **RETURNS** clause specifies the return type for the function.
 - The **AS** clause specifies the class name and static method name of the Java method to be invoked. If the **NULL** value needs to be transferred to the Java method as an input parameter, specify the name of the Java encapsulation class corresponding to the parameter type. For details, see [UDF Example: Processing NULL Values](#).
 - For details, see section "CREATE FUNCTION" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

```
CREATE [ OR REPLACE ] FUNCTION function_name
( [ { argname [ argmode ] argtype [ { DEFAULT | := | = } expression ] } [, ...] ] )
[ RETURNS rettype [ DETERMINISTIC ] ]
LANGUAGE JAVA
[
  { IMMUTABLE | STABLE | VOLATILE }
  | [ NOT ] LEAKPROOF
  | WINDOW
  | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
  | [ { EXTERNAL } SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER | AUTHID DEFINER |
    AUTHID CURRENT_USER }
  | { FENCED }
  | COST execution_cost
  | ROWS result_rows
  | SET configuration_parameter { { TO |= } value | FROM CURRENT }
] [...]
{
  AS 'class_name.method_name' ( { argtype } [, ...] )
}
```
- Use functions.**
 During execution, PL/Java searches for the Java class specified by a function among all the deployed JAR packages, which are ranked by name in

alphabetical order, invokes the Java method in the first found class, and returns results.

- Delete functions.

PL/Java functions can be deleted by using the **DROP FUNCTION** syntax.

```
DROP FUNCTION [ IF EXISTS ] function_name ( ( [ { [ argmode ] [ argname ] argtype } [, ...] ] )  
[ CASCADE | RESTRICT ] );
```

To delete an overloaded function (for details, see [UDF Example: Overloaded Functions](#)), specify **argtype** in the function. To delete other functions, simply specify **function_name**.

- Authorize permissions for functions.

Only user **sysadmin** can create PL/Java functions. It can also grant other users the permission to use the PL/Java functions. For details, see "GRANT" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }  
ON { FUNCTION {function_name ( [ { [ argmode ] [ arg_name ] arg_type } [, ...] ] ) } [, ...]  
| ALL FUNCTIONS IN SCHEMA schema_name [, ...] }  
TO { [ GROUP ] role_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];
```

UDF Example: Processing Array Types

DWS can convert basic array types. You only need to append a pair of square brackets ([]) to the data type when creating a function.

```
CREATE FUNCTION java_arrayLength(INTEGER[])  
RETURNS INTEGER  
AS 'Example.getArrayLength'  
LANGUAGE JAVA;
```

Java code is similar to the following:

```
public class Example  
{  
    public static int getArrayLength(Integer[] intArray)  
    {  
        return intArray.length;  
    }  
}
```

Invoke the following statement:

```
SELECT java_arrayLength(ARRAY[1, 2, 3]);
```

The expected result is as follows:

```
java_arrayLength  
-----  
3  
(1 row)
```

UDF Example: Processing NULL Values

NULL values cannot be handled for DWS data types that are mapped and can be converted to simple Java types by default. If you use a Java function to obtain and process the **NULL** value transferred from DWS, specify the Java encapsulation class in the **AS** clause as follows:

```
CREATE FUNCTION java_countnulls(INTEGER[])  
RETURNS INTEGER  
AS 'Example.countNulls(java.lang.Integer[])'  
LANGUAGE JAVA;
```


Java code is similar to the following:

```
public class Example
{
    public static int countNulls(Integer[] intArray)
    {
        int nullCount = 0;
        for (int idx = 0; idx < intArray.length; ++idx)
        {
            if (intArray[idx] == null)
                nullCount++;
        }
        return nullCount;
    }
}
```

Invoke the following statement:

```
SELECT java_countNulls(ARRAY[null, 1, null, 2, null]);
```

The expected result is as follows:

```
java_countNulls
-----
3
(1 row)
```

UDF Example: Overloaded Functions

PL/Java supports overloaded functions. You can create functions with the same name or invoke overloaded functions from Java code. The procedure is as follows:

Step 1 Create overloaded functions.

For example, create two Java methods with the same name, and specify the methods `dummy(int)` and `dummy(String)` with different parameter types.

```
public class Example
{
    public static int dummy(int value)
    {
        return value*2;
    }
    public static String dummy(String value)
    {
        return value;
    }
}
```

In addition, create two functions with the same names as the above two functions in DWS.

```
CREATE FUNCTION java_dummy(INTEGER)
    RETURNS INTEGER
    AS 'Example.dummy'
LANGUAGE JAVA;

CREATE FUNCTION java_dummy(VARCHAR)
    RETURNS VARCHAR
    AS 'Example.dummy'
LANGUAGE JAVA;
```

Step 2 Invoke the overloaded functions.

DWS invokes the functions that match the specified parameter type. The results of invoking the above two functions are as follows:

```
SELECT java_dummy(5);
java_dummy
-----
10
(1 row)
```

```
SELECT java_dummy('5');
java_dummy
-----
5
(1 row)
```

Note that DWS may implicitly convert data types. Therefore, you are advised to specify the parameter type when invoking an overloaded function.

```
SELECT java_dummy(5::varchar);
java_dummy
-----
5
(1 row)
```

In this case, the specified parameter type is preferentially used for matching. If there is no Java method matching the specified parameter type, the system implicitly converts the parameter and searches for Java methods based on the conversion result.

```
SELECT java_dummy(5::INTEGER);
java_dummy
-----
10
(1 row)

DROP FUNCTION java_dummy(INTEGER);

SELECT java_dummy(5::INTEGER);
java_dummy
-----
5
(1 row)
```

NOTICE

Data types supporting implicit conversion are as follows:

- **SMALLINT**: It can be converted to the **INTEGER** type by default.
- **SMALLINT** and **INTEGER**: They can be converted to the **BIGINT** type by default.
- **TINYINT**, **SMALLINT**, **INTEGER**, and **BIGINT**: They can be converted to the **BOOL** type by default.
- The following data types can be converted to **TEXT** by default: **CHAR**, **NAME**, **BIGINT**, **INTEGER**, **SMALLINT**, **TINYINT**, **RAW**, **FLOAT4**, **FLOAT8**, **BPCHAR**, **VARCHAR**, **NVARCHAR2**, **DATE**, **TIMESTAMP**, **TIMESTAMPTZ**, **NUMERIC**, and **SMALLDATETIME**.
- The following data types can be converted to **VARCHAR** by default: **TEXT**, **CHAR**, **BIGINT**, **INTEGER**, **SMALLINT**, **TINYINT**, **RAW**, **FLOAT4**, **FLOAT8**, **BPCHAR**, **DATE**, **NVARCHAR2**, **TIMESTAMP**, **NUMERIC**, and **SMALLDATETIME**.

Step 3 Delete the overloaded functions.

To delete an overloaded function, specify the parameter type for the function. Otherwise, the function cannot be deleted.

```
DROP FUNCTION java_dummy(INTEGER);
```

----End

UDF-related GUC Parameters

- **udf_memory_limit**

A system-level GUC parameter. It is used to limit the physical memory used by each CN or DN for executing UDFs. The default value is **0.05 * max_process_memory**. You can use the **postgresql.conf** file to modify the parameter setting. The modification takes effect only after the database is restarted.

NOTICE

- **udf_memory_limit** is a part of **max_process_memory**. When a CN or DN is started, memory calculated by **udf_memory_limit** minus **200 MB** will be reserved for UDF Worker processes. CN and DN processes are different from the UDF Worker process, and the CN and DN processes will save memory for the UDF Worker process.

For example, if **max_process_memory** is set to **10GB** on a DN and **udf_memory_limit** is set to **4GB**, the DN can use a maximum of 6.2 GB memory, that is, 10 GB – (4 GB – 200 MB). This case applies even if no UDF is executed. By default, the value of **udf_memory_limit** is **0.05 * max_process_memory**. Querying the **pv_total_memory_detail** view will prove that the value of **process_used_memory** would never exceed the calculation result of **max_process_memory – (udf_memory_limit – 200 MB)**.

- If the UDF process is disconnected, an error message will be displayed. Example: "memory in UDF Work Process is limited by cgroup: [usage: xxx, max_usage_history: xxx, limit: xxx]." You can learn the current memory usage from this message. In the error information, **usage** indicates the total physical memory used by the rest of the UDF process after a UDF process is killed. **max_usage_history** indicates the highest memory usage of the UDF process after the UDF instance is started. **limit** indicates the maximum memory used by the UDF process. If the value of **max_usage_history** is close to the value of **limit**, the memory usage of the current cluster may exceed the limit. In this case, optimize workloads or adjust the value of **udf_memory_limit** as needed.
- Executing a simplest Java UDF on a CN consumes about 50 MB physical memory. You can set this parameter based on the memory usage and concurrency of Java functions to be used. After this parameter is added, you are not advised to set **UDFWorkerMemHardLimit** and **FencedUDFMemoryLimit**.
- If the parallelism of the UDF process is excessively high and the memory usage exceeds the **udf_memory_limit** value, unexpected situations such as process exit may occur. In this scenario, the execution result may be unreliable. You are advised to set this parameter to reserve sufficient memory based on the site requirements. If the system has the **/var/log/messages** log, check the log to see whether the memory is insufficient because the cgroup memory limit has been reached. If the memory is severely insufficient, the UDF master process may exit. You can view the UDF log for analysis. The default UDF log path is **\$GAUSSLOG/cm/cm_agent/pg_log**. For example, if the log below is displayed, the memory resources are insufficient and the UDF master process exits. In this case, you need to check the **udf_memory_limit** parameter.

0 [BACKEND] FATAL: poll() failed: Bad address, please check the parameter:udf_memory_limit to make sure there is enough memory.

- **FencedUDFMemoryLimit**

A session-level GUC parameter. It is used to specify the maximum virtual memory used by a single Fenced UDF Worker process initiated by a session.

```
SET FencedUDFMemoryLimit='512MB';
```

The value range of this parameter is (**150 MB**, **1G**). If the value is greater than **1G**, an error will be reported immediately. If the value is less than or equal to **150 MB**, an error will be reported during function invoking.

NOTICE

- If **FencedUDFMemoryLimit** is set to **0**, the virtual memory for a Fenced UDF Worker process will not be limited.
- You are advised to use **udf_memory_limit** to control the physical memory used by Fenced UDF Worker processes. You are not advised to use **FencedUDFMemoryLimit**, especially when Java UDFs are used. If you are clear about the impact of this parameter, set it based on the following information:
 - After a C Fenced UDF Worker process is started, it will occupy about 200 MB virtual memory, and about 16 MB physical memory.
 - After a Java Fenced UDF Worker process is started, it will occupy about 2.5 GB virtual memory, and about 50 MB physical memory.

Exception Handling

If there is an exception in a JVM, PL/Java will export JVM stack information during the exception to a client.

Logging

PL/Java uses the standard Java Logger. Therefore, you can record logs as follows:

```
Logger.getAnonymousLogger().config( "Time is " + new  
Date(System.currentTimeMillis()));
```

An initialized Java Logger class is set to the **CONFIG** level by default, corresponding to the **LOG** level in DWS. In this case, log messages generated by Java Logger are all redirected to the DWS backend. Then, the log messages are written into server logs or displayed on the user interface. MPPDB server logs record information at the **LOG**, **WARNING**, and **ERROR** levels. The SQL user interface displays logs at the **WARNING** and **ERROR** levels. The following table lists mapping between Java Logger levels and DWS log levels.

Table 10-3 PL/Java log levels

java.util.logging.Level	DWS Log Level
SERVER	ERROR
WARNING	WARNING
CONFIG	LOG
INFO	INFO
FINE	DEBUG1
FINER	DEBUG2

java.util.logging.Level	DWS Log Level
FINEST	DEBUG3

You can change Java Logger levels. For example, if the Java Logger level is changed to **SEVERE** by the following Java code, log messages (**msg**) will not be recorded in DWS logs during **WARNING** logging.

```
Logger log = Logger.getAnonymousLogger();  
Log.setLevel(Level.SEVERE);  
log.log(Level.WARNING, msg);
```

Security Issues

In DWS, PL/Java is an untrusted language. Only user **sysadmin** can create PL/Java functions. The user can grant other users the permission for using the PL/Java functions. For details, see [Authorize permissions for functions](#).

In addition, PL/Java controls user access to file systems, forbidding users from reading most system files, or writing, deleting, or executing any system files in Java methods.

10.2 DWS PL/pgSQL Functions

PL/pgSQL is similar to PL/SQL of Oracle. It is a loadable procedural language.

The functions created using PL/pgSQL can be used in any place where you can use built-in functions. For example, you can create calculation functions with complex conditions and use them to define operators or use them for index expressions.

SQL is used by most databases as a query language. It is portable and easy to learn. Each SQL statement must be executed independently by a database server.

In this case, when a client application sends a query to the server, it must wait for it to be processed, receive and process the results, and then perform some calculation before sending more queries to the server. If the client and server are not on the same machine, all these operations will cause inter-process communication and increase network loads.

PL/pgSQL 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/pgSQL can use all data types, operators, and functions in SQL.

For details about the PL/pgSQL syntax for creating functions, see **CREATE FUNCTION**. As mentioned earlier, PL/pgSQL is similar to PL/SQL of Oracle and is a loadable procedural language. Its application method is similar to that of **DWS**

Stored Procedures. There is only one difference. Stored procedures have no return values but the functions have.

11 DWS Stored Procedures

11.1 Overview

What Is a DWS Stored Procedure?

In DWS, business rules and logics are saved as stored procedures.

A stored procedure is a combination of SQL, PL/SQL, and Java statements. Stored procedures can move the code that executes business rules from applications to databases. In this way, code can be used by multiple programs at a time.

For how to create and invoke a stored procedure, see "CREATE PROCEDURE" in the *Data Warehouse Service (DWS) SQL Syntax Reference*.

The functions created using the PL/pgSQL language mentioned in [DWS PL/pgSQL Functions](#) are similar to the application methods of stored procedures. Unless otherwise specified, the following sections apply to stored procedures and PL/pgSQL functions.

DWS Stored Procedure Data Types

A data type refers to a value set and an operation set defined on the value set. A DWS database consists of tables, each of which is defined by its own columns. Each column corresponds to a data type. DWS uses corresponding functions to perform operations on data based on data types. For example, DWS can perform addition, subtraction, multiplication, and division operations on data of numeric values.

11.2 Data Type Conversion for DWS Stored Procedures

Certain data types in the database support implicit data type conversions, such as assignments and parameters invoked by functions. For other data types, you can use the type conversion functions provided by DWS, such as the CAST function, to forcibly convert them.

[Table 11-1](#) lists common implicit data type conversions in DWS.

NOTICE

The valid value range of DATE supported by DWS is from 4713 B.C. to 294276 A.D.

Table 11-1 Implicit data type conversions

Raw Data Type	Target Data Type	Remarks
CHAR	VARCHAR2	N/A
CHAR	NUMBER	Raw data must consist of digits.
CHAR	DATE	Raw data cannot exceed the valid date range.
CHAR	RAW	N/A
CHAR	CLOB	N/A
VARCHAR2	CHAR	N/A
VARCHAR2	NUMBER	Raw data must consist of digits.
VARCHAR2	DATE	Raw data cannot exceed the valid date range.
VARCHAR2	CLOB	N/A
NUMBER	CHAR	N/A
NUMBER	VARCHAR2	N/A
DATE	CHAR	N/A
DATE	VARCHAR2	N/A
RAW	CHAR	N/A
RAW	VARCHAR2	N/A
CLOB	CHAR	N/A
CLOB	VARCHAR2	N/A
CLOB	NUMBER	Raw data must consist of digits.
INT4	CHAR	N/A

11.3 DWS Stored Procedure Array and Record

11.3.1 Arrays

Use of Array Types

Before the use of arrays, an array type needs to be defined:

Define an array type immediately after the **AS** keyword in a stored procedure. Run the following statement:

```
TYPE array_type IS VARRAY(size) OF data_type [NOT NULL];
```

Its parameters are as follows:

- **array_type**: indicates the name of the array type to be defined.
- **VARRAY**: indicates the array type to be defined.
- **size**: indicates the maximum number of members in the array type to be defined. The value is a positive integer.
- **data_type**: indicates the types of members in the array type to be created.
- **NOT NULL**: an optional constraint. It can be used to ensure that none of the elements in the array is **NULL**.

NOTE

- In DWS, an array automatically increases. If an access violation occurs, a null value will be returned, and no error message will be reported. If out-of-bounds write occurs in an array, the message **Subscript outside of limit** is displayed.
- The scope of an array type defined in a stored procedure takes effect only in this storage process.
- It is recommended that you use one of the preceding methods to define an array type. If both methods are used to define the same array type, DWS prefers the array type defined in a stored procedure to declare array variables.

In DWS 8.1.0 and earlier versions, the system does not verify the length of array elements and out-of-bounds write because the array can automatically increase. This version adds related constraints to be compatible with Oracle databases. If out-of-bounds write exists, you can configure **varray_verification** in the parameter **behavior_compat_options** to be compatible with previously unverified operations.

Example:

```
-- Declare an array in a stored procedure.
CREATE OR REPLACE PROCEDURE array_proc
AS
    TYPE ARRAY_INTEGER IS VARRAY(1024) OF INTEGER;--Define the array type.
    TYPE ARRAY_INTEGER_NOT_NULL IS VARRAY(1024) OF INTEGER NOT NULL;-- Defines non-null array
types.
    ARRINT ARRAY_INTEGER := ARRAY_INTEGER(); --Declare the variable of the array type.
BEGIN
    ARRINT.extend(10);
    FOR I IN 1..10 LOOP
        ARRINT(I) := I;
    END LOOP;
    DBMS_OUTPUT.PUT_LINE(ARRINT.COUNT);
    DBMS_OUTPUT.PUT_LINE(ARRINT(1));
    DBMS_OUTPUT.PUT_LINE(ARRINT(10));
    DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.FIRST));
    DBMS_OUTPUT.PUT_LINE(ARRINT(ARRINT.last));
END;
/
```

```
-- Invoke the stored procedure.  
CALL array_proc();  
10  
1  
10  
1  
10  
-- Delete the stored procedure.  
DROP PROCEDURE array_proc;
```

Declaration and Use of Rowtype Arrays

In addition to the declaration and use of common arrays and non-null arrays in the preceding example, the array also supports the declaration and use of rowtype arrays.

Example:

```
-- Use the COUNT function on an array in a stored procedure.  
CREATE TABLE tbl (a int, b int);  
INSERT INTO tbl VALUES(1, 2),(2, 3),(3, 4);  
CREATE OR REPLACE PROCEDURE array_proc  
AS  
    CURSOR all_tbl IS SELECT * FROM tbl ORDER BY a;  
    TYPE tbl_array_type IS varray(50) OF tbl%rowtype; -- Defines the array of the rowtype type. tbl indicates  
any table.  
    tbl_array tbl_array_type;  
    tbl_item tbl%rowtype;  
    inx1 int;  
BEGIN  
    tbl_array := tbl_array_type();  
    inx1 := 0;  
    FOR tbl_item IN all_tbl LOOP  
        inx1 := inx1 + 1;  
        tbl_array(inx1) := tbl_item;  
    END LOOP;  
    WHILE inx1 IS NOT NULL LOOP  
        DBMS_OUTPUT.PUT_LINE('tbl_array(inx1).a=' || tbl_array(inx1).a || ' tbl_array(inx1).b=' ||  
tbl_array(inx1).b);  
        inx1 := tbl_array.PRIOR(inx1);  
    END LOOP;  
END;  
/
```

The execution output is as follows:

```
call array_proc();  
tbl_array(inx1).a=3 tbl_array(inx1).b=4  
tbl_array(inx1).a=2 tbl_array(inx1).b=3  
tbl_array(inx1).a=1 tbl_array(inx1).b=2
```

Array Related Functions

DWS supports Oracle-related array functions. You can use the following functions to obtain array attributes or perform operations on the array content.

COUNT

Returns the number of elements in the current array. Only the initialized elements or the elements extended by the EXTEND function are counted.

Use:

varray.**COUNT** or *varray*.**COUNT()**

Example:

```
-- Use the COUNT function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
    v_varray.extend;
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count=3
v_varray.count=4
```

FIRST and LAST

The FIRST function can return the subscript of the first element. The LAST function can return the subscript of the last element.

Use:

varray.FIRST or *varray*.FIRST()

varray.LAST or *varray*.LAST()

Example:

```
-- Use the FIRST and LAST functions on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    DBMS_OUTPUT.PUT_LINE('v_varray.first=' || v_varray.first);
    DBMS_OUTPUT.PUT_LINE('v_varray.last=' || v_varray.last);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.first=1
v_varray.last=3
```

EXTEND

NOTE

The EXTEND function is used to be compatible with two Oracle database operations. In DWS, an array automatically grows, and the EXTEND function is not necessary. For a newly written stored procedure, you do not need to use the EXTEND function.

The EXTEND function can extend arrays. The EXTEND function can be invoked in either of the following ways:

- Method 1:

EXTEND contains an integer input parameter, indicating that the array size is extended by the specified length. When the EXTEND function is executed, the

values returned by the COUNT and LAST functions will be updated accordingly.

Use:

varray.EXTEND(size)

By default, one bit is added to the end of *varray*.EXTEND, which is equivalent to *varray*.EXTEND(1).

- Method 2:

EXTEND contains two integer input parameters. The first parameter indicates the length of the extended size. The second parameter indicates that the value of the extended array element is the same as that of the element with the **index** subscript.

Use:

varray.EXTEND(size, index)

Example:

```
-- Use the EXTEND function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    v_varray.extend(3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
    v_varray.extend(2,3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count=' || v_varray.count);
    DBMS_OUTPUT.PUT_LINE('v_varray(7)= ' || v_varray(7));
    DBMS_OUTPUT.PUT_LINE('v_varray(8)= ' || v_varray(7));
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count=6
v_varray.count=8
v_varray(7)=3
v_varray(8)=3
```

NEXT and PRIOR

The NEXT and PRIOR functions are used for cyclic array traversal. The NEXT function returns the subscript of the next array element based on the input parameter **index**. If the subscript reaches the maximum value, **NULL** is returned. The PRIOR function returns the subscript of the previous array element based on the input parameter **index**. If the minimum value of the array subscript is reached, **NULL** is returned.

Use:

varray.NEXT(index)

varray.PRIOR(index)

Example:

```
-- Use the NEXT and PRIOR functions on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
```

```
TYPE varray_type IS VARRAY(20) OF INT;
v_varray varray_type;
i int;
BEGIN
    v_varray := varray_type(1, 2, 3);

    i := v_varray.COUNT;
    WHILE i IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE('test prior v_varray('||i||')=' || v_varray(i));
        i := v_varray.PRIOR(i);
    END LOOP;

    i := 1;
    WHILE i IS NOT NULL LOOP
        DBMS_OUTPUT.PUT_LINE('test next v_varray('||i||')=' || v_varray(i));
        i := v_varray.NEXT(i);
    END LOOP;
END;
/
```

The execution output is as follows:

```
call test_varray();
test prior v_varray(3)=3
test prior v_varray(2)=2
test prior v_varray(1)=1
test next v_varray(1)=1
test next v_varray(2)=2
test next v_varray(3)=3
```

EXISTS

Determines whether an array subscript exists.

Use:

varray.EXISTS(index)

Example:

```
-- Use the EXISTS function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3);
    IF v_varray.EXISTS(1) THEN
        DBMS_OUTPUT.PUT_LINE('v_varray.EXISTS(1)');
    END IF;
    IF NOT v_varray.EXISTS(10) THEN
        DBMS_OUTPUT.PUT_LINE('NOT v_varray.EXISTS(10)');
    END IF;
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.EXISTS(1)
NOT v_varray.EXISTS(10)
```

TRIM

Deletes a specified number of elements from the end of an array.

Use:

varray.TRIM(size)

varray.TRIM is equivalent to *varray*.TRIM(1), because the default input parameter is 1.

Example:

```
-- Use the TRIM function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3, 4, 5);
    v_varray.trim(3);
    DBMS_OUTPUT.PUT_LINE('v_varray.count' || v_varray.count);
    v_varray.trim;
    DBMS_OUTPUT.PUT_LINE('v_varray.count:' || v_varray.count);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count:2
v_varray.count:1
```

DELETE

Deletes all elements from an array.

Use:

varray.DELETE or *varray*.DELETE()

Example:

```
-- Use the DELETE function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
AS
    TYPE varray_type IS VARRAY(20) OF INT;
    v_varray varray_type;
BEGIN
    v_varray := varray_type(1, 2, 3, 4, 5);
    v_varray.delete;
    DBMS_OUTPUT.PUT_LINE('v_varray.count:' || v_varray.count);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.count:0
```

LIMIT

Returns the allowed maximum length of an array.

Use:

varray.LIMIT or *varray*.LIMIT()

Example:

```
-- Use the LIMIT function on an array in a stored procedure.
CREATE OR REPLACE PROCEDURE test_varray
```

```
AS
  TYPE varray_type IS VARRAY(20) OF INT;
  v_varray varray_type;
BEGIN
  v_varray := varray_type(1, 2, 3, 4, 5);
  DBMS_OUTPUT.PUT_LINE('v_varray.limit:' || v_varray.limit);
END;
/
```

The execution output is as follows:

```
call test_varray();
v_varray.limit:20
```

11.3.2 record

record Variables

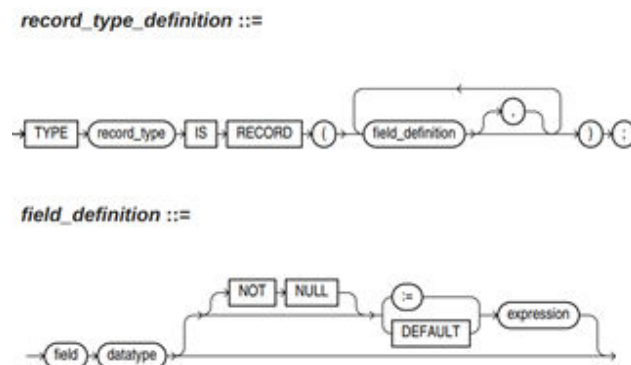
Perform the following operations to create a record variable:

Define a record type and use this type to declare a variable.

Syntax

For the syntax of the record type, see [Figure 11-1](#).

Figure 11-1 Syntax of the record type



The syntax is described as follows:

- **record_type**: record name
- **field**: record columns
- **datatype**: record data type
- **expression**: expression for setting a default value

 NOTE

In DWS:

- When assigning values to record variables, you can:
 - Declare a record type and define member variables of this type when you declare a function or stored procedure.
 - Assign the value of a record variable to another record variable.
 - Use **SELECT INTO** or **FETCH** to assign values to a record type.
 - Assign the **NULL** value to a record variable.
- The **INSERT** and **UPDATE** statements cannot use a record variable to insert or update data.
- Just like a variable, a record column of the compound type does not have a default value in the declaration.

Examples

The table used in the following stored procedure is defined as follows:

```
CREATE TABLE emp_rec
```

```
(
  empno      numeric(4,0),
  ename      character varying(10),
  job        character varying(9),
  mgr        numeric(4,0),
  hiredate   timestamp(0) without time zone,
  sal        numeric(7,2),
  comm       numeric(7,2),
  deptno     numeric(2,0)
)
```

```
with (orientation = column,compression=middle)
```

```
distribute by hash (sal);
```

```
\d emp_rec
```

Table "public.emp_rec"			
Column	Type	Modifiers	
empno	numeric(4,0)	not null	
ename	character varying(10)		
job	character varying(9)		
mgr	numeric(4,0)		
hiredate	timestamp(0) without time zone		
sal	numeric(7,2)		
comm	numeric(7,2)		
deptno	numeric(2,0)		

```
-- Perform array operations in the stored procedure.
```

```
CREATE OR REPLACE FUNCTION regress_record(p_w VARCHAR2)
```

```
RETURNS
```

```
VARCHAR2 AS $$
```

```
DECLARE
```

```
-- Declare a record type.
```

```
type rec_type is record (name varchar2(100), epno int);
```

```
employer rec_type;
```

```
-- Use %type to declare the record type.
```

```
type rec_type1 is record (name emp_rec.ename%type, epno int not null :=10);
```

```
employer1 rec_type1;
```

```
-- Declare a record type with a default value.
```

```
type rec_type2 is record (
```

```
  name varchar2 not null := 'SCOTT',
```

```
  epno int not null :=10);
```

```
employer2 rec_type2;
```

```
CURSOR C1 IS select ename,epno from emp_rec order by 1 limit 1;
```

```
BEGIN
  -- Assign a value to a member record variable.
  employer.name := 'WARD';
  employer.epno = 18;
  raise info 'employer name: % , epno:%', employer.name, employer.epno;

  -- Assign the value of a record variable to another variable.
  employer1 := employer;
  raise info 'employer1 name: % , epno: %', employer1.name, employer1.epno;

  -- Assign the NULL value to a record variable.
  employer1 := NULL;
  raise info 'employer1 name: % , epno: %', employer1.name, employer1.epno;

  -- Obtain the default value of a record variable.
  raise info 'employer2 name: % ,epno: %', employer2.name, employer2.epno;

  -- Use a record variable in the FOR loop.
  for employer in select ename,empno from emp_rec order by 1 limit 1
  loop
    raise info 'employer name: % , epno: %', employer.name, employer.epno;
  end loop;

  -- Use a record variable in the SELECT INTO statement.
  select ename,empno into employer2 from emp_rec order by 1 limit 1;
  raise info 'employer name: % , epno: %', employer2.name, employer2.epno;

  -- Use a record variable in a cursor.
  OPEN C1;
  FETCH C1 INTO employer2;
  raise info 'employer name: % , epno: %', employer2.name, employer2.epno;
  CLOSE C1;
  RETURN employer.name;
END;
$$
LANGUAGE plpgsql;

-- Invoke the stored procedure.
CALL regress_record('abc');
INFO: employer name: WARD , epno:18
INFO: employer1 name: WARD , epno: 18
INFO: employer1 name: <NULL> , epno: <NULL>
INFO: employer2 name: SCOTT ,epno: 10
-- Delete the stored procedure.
DROP PROCEDURE regress_record;
```

11.4 Declaration Syntax of DWS Stored Procedures

Basic Structure

A PL/SQL block can contain a sub-block which can be placed in any section. The following describes the architecture of a PL/SQL block:

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

DECLARE

NOTE

This part is optional if no variable needs to be declared.

- An anonymous block may omit the **DECLARE** keyword if no variable needs to be declared.
- For a stored procedure, **AS** is used, which is equivalent to **DECLARE**. The **AS** keyword must be reserved even if there is no variable declaration part.

- **EXECUTION:** specifies procedure and SQL statements. It is the main part of a program. It is mandatory.
BEGIN
- **EXCEPTION:** processes errors. It is optional.
EXCEPTION
- **END**
END;
/

NOTICE

You are not allowed to use consecutive tabs in the PL/SQL block, because they may result in an exception when the parameter **-r** is executed using the **gsql** tool.

PL/SQL Block Classification

PL/SQL blocks are classified into the following types:

- Anonymous block: a dynamic block that can be executed only for once. For details about the syntax, see [Anonymous Block](#).
- Subprogram: a stored procedure, function, operator, or packages stored in a database. A subprogram created in a database can be called by other programs.

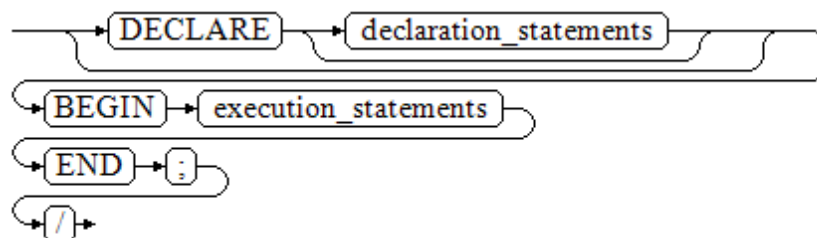
Anonymous Block

An anonymous block applies to a script infrequently executed or a one-off activity. An anonymous block is executed in a session and is not stored.

Syntax

[Figure 11-2](#) shows the syntax diagrams for an anonymous block.

Figure 11-2 anonymous_block::=



Details about the syntax diagram are as follows:

- The execute part of an anonymous block starts with a **BEGIN** statement, has a break with an **END** statement, and ends with a semicolon (;). Type a slash (/) and press **Enter** to execute the statement.

NOTICE

The terminator "/" must be written in an independent row.

- The declaration section includes the variable definition, type, and cursor definition.
- A simplest anonymous block does not execute any commands. At least one statement, even a null statement, must be presented in any implementation blocks.

Examples

The following lists basic anonymous block programs:

```
-- Null statement block:
BEGIN
  NULL;
END;
/

-- Print information to the console:
BEGIN
  dbms_output.put_line('hello world!');
END;
/

-- Print variable contents to the console:
DECLARE
  my_var VARCHAR2(30);
BEGIN
  my_var := 'world';
  dbms_output.put_line('hello'||my_var);
END;
/
```

Subprogram

A subprogram stores stored procedures, functions, operators, and advanced packages. A subprogram created in a database can be called by other programs.

11.5 Basic Statements of DWS Stored Procedures

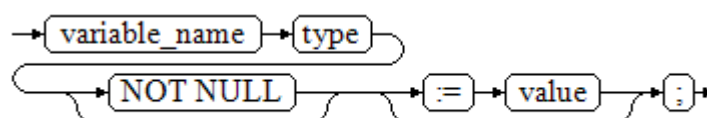
Variable Definition Statement

This section describes the declaration of variables in the PL/SQL and the scope of this variable in codes.

Variable declaration

For details about the variable declaration syntax, see [Figure 11-3](#).

Figure 11-3 declare_variable::=



The syntax is described as follows:

- **variable_name** indicates the name of a variable.
- **type** indicates the type of a variable.
- **value** indicates the initial value of the variable. (If the initial value is not given, **NULL** is taken as the initial value.) **value** can also be an expression.

Examples

```
DECLARE
  emp_id INTEGER := 7788; -- Define a variable and assign a value to it.
BEGIN
  emp_id := 5*7784; -- Assign a value to the variable.
END;
```

In addition to the declaration of basic variable types, **%TYPE** and **%ROWTYPE** can be used to declare variables related to table columns or table structures.

%TYPE attribute

%TYPE declares a variable to be of the same data type as a previously declared variable (for example, a column in a table). For example, if you want to define a **my_name** variable that has the same data type as the **firstname** column in the **employee** table, you can define the variable as follows:

```
my_name employee.firstname%TYPE
```

In this way, you can declare **my_name** even if you do not know the data type of **firstname** in **employee**, and the data type of **my_name** can be automatically updated when the data type of **firstname** changes.

%ROWTYPE attribute

%ROWTYPE declares data types of a set of data. It stores a row of table data or results fetched from a cursor. For example, if you want to define a set of data with the same column names and column data types as the **employee** table, you can define the data as follows:

```
my_employee employee%ROWTYPE
```

NOTICE

If multiple CNs are used, the **%ROWTYPE** and **%TYPE** attributes of temporary tables cannot be declared in a stored procedure, because a temporary table is valid only in the current session and is invisible to other CNs in the compilation phase. In this case, a message is displayed indicating that the temporary table does not exist.

Variable scope

The scope of a variable indicates the accessibility and availability of a variable in code block. In other words, a variable takes effect only within its scope.

- To define a function scope, a variable must declare and create a **BEGIN-END** block in the declaration section. The necessity of such declaration is also determined by block structure, which requires that a variable has different scopes and lifetime during a process.

- A variable can be defined multiple times in different scopes, and inner definition can cover outer one.
- A variable defined in an outer block can also be used in a nested block. However, the outer block cannot access variables in the nested block.

Examples

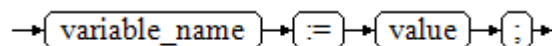
```
DECLARE
  emp_id INTEGER :=7788; -- Define a variable and assign a value to it.
  outer_var INTEGER :=6688; -- Define a variable and assign a value to it.
BEGIN
  DECLARE
    emp_id INTEGER :=7799; -- Define a variable and assign a value to it.
    inner_var INTEGER :=6688; -- Define a variable and assign a value to it.
  BEGIN
    dbms_output.put_line('inner emp_id ='||emp_id); -- Display the value as 7799.
    dbms_output.put_line('outer_var ='||outer_var); -- Cite variables of an outer block.
  END;
  dbms_output.put_line('outer emp_id ='||emp_id); -- Display the value as 7788.
END;
/
```

Assignment Statement

Syntax

Figure 11-4 shows the syntax diagram for assigning a value to a variable.

Figure 11-4 assignment_value::=



The syntax is described as follows:

- **variable_name** indicates the name of a variable.
- **value** can be a value or an expression. The type of **value** must be compatible with the type of **variable_name**.

Examples

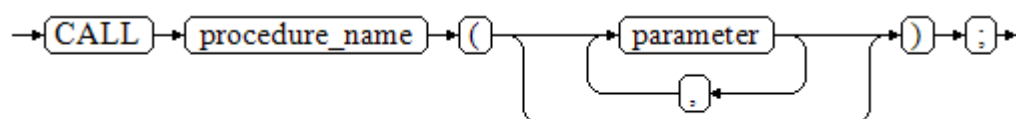
```
DECLARE
  emp_id INTEGER := 7788; --Assignment
BEGIN
  emp_id := 5; --Assignment
  emp_id := 5*7784;
END;
/
```

Call Statement

Syntax

Figure 11-5 shows the syntax diagram for calling a clause.

Figure 11-5 call_clause::=



The syntax is described as follows:

- **procedure_name** specifies the name of a stored procedure.
- **parameter** specifies the parameters for the stored procedure. You can set no parameter or multiple parameters.

Examples

```
-- Create the stored procedure proc_staffs:
CREATE OR REPLACE PROCEDURE proc_staffs
(
  section    NUMBER(6),
  salary_sum out NUMBER(8,2),
  staffs_count out INTEGER
)
IS
BEGIN
  SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM staffs where section_id = section;
END;
/

-- Create the stored procedure proc_return:
CREATE OR REPLACE PROCEDURE proc_return
AS
  v_num NUMBER(8,2);
  v_sum INTEGER;
BEGIN
  proc_staffs(30, v_sum, v_num); --Invoke a statement:
  dbms_output.put_line(v_sum||'#'||v_num);
  RETURN; --Return a statement.
END;
/

-- Invoke a stored procedure proc_return:
CALL proc_return();

-- Delete a stored procedure:
DROP PROCEDURE proc_staffs;
DROP PROCEDURE proc_return;

--Create the function func_return.
CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
  v_num INTEGER := 1;
BEGIN
  dbms_output.put_line(v_num);
  RETURN; --Return a statement.
END $$;

-- Invoke the function func_return.
CALL func_return();
1

-- Delete the function:
DROP FUNCTION func_return;
```

11.6 Dynamic Statements of DWS Stored Procedures

11.6.1 Executing Dynamic Query Statements

You can perform dynamic queries using **EXECUTE IMMEDIATE** or **OPEN FOR** in DWS. **EXECUTE IMMEDIATE** dynamically executes **SELECT** statements and **OPEN**

FOR combines use of cursors. If you need to store query results in a data set, use **OPEN FOR**.

EXECUTE IMMEDIATE

Figure 11-6 shows the syntax diagram.

Figure 11-6 EXECUTE IMMEDIATE dynamic_select_clause::=

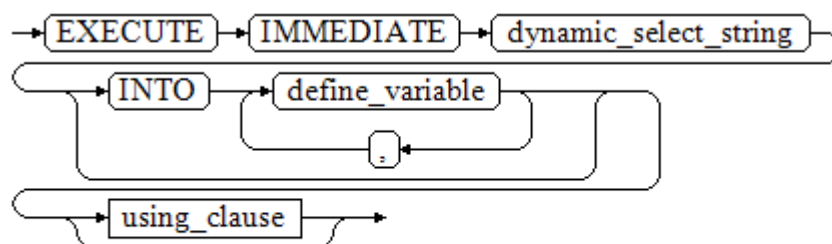
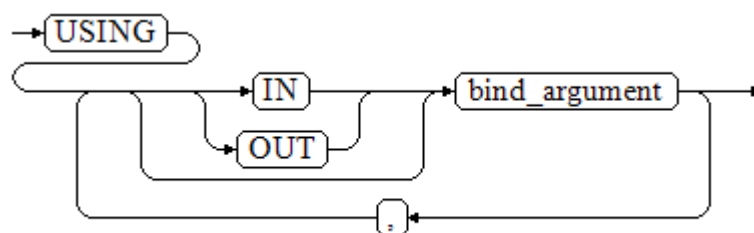


Figure 11-7 shows the syntax diagram for **using_clause**.

Figure 11-7 using_clause-1



The above syntax diagram is explained as follows:

- **define_variable**: specifies variables to store single-line query results.
- **USING IN bind_argument**: specifies where the variable passed to the dynamic SQL value is stored, that is, in the dynamic placeholder of **dynamic_select_string**.
- **USING OUT bind_argument**: specifies where the dynamic SQL returns the value of the variable.

NOTICE

- In query statements, **INTO** and **OUT** cannot coexist.
- A placeholder name starts with a colon (:) followed by digits, characters, or strings, corresponding to *bind_argument* in the **USING** clause.
- *bind_argument* can only be a value, variable, or expression. It cannot be a database object such as a table name, column name, and data type. That is, *bind_argument* cannot be used to transfer schema objects for dynamic SQL statements. If a stored procedure needs to transfer database objects through *bind_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic_select_clause* with a database object.
- A dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind_argument* in the **USING** clause.

Example

```
--Retrieve values from dynamic statements (INTO clause).
DECLARE
  staff_count VARCHAR2(20);
BEGIN
  EXECUTE IMMEDIATE 'select count(*) from staffs'
    INTO staff_count;
  dbms_output.put_line(staff_count);
END;
/

--Pass and retrieve values (the INTO clause is used before the USING clause).
CREATE OR REPLACE PROCEDURE dynamic_proc
AS
  staff_id    NUMBER(6) := 200;
  first_name  VARCHAR2(20);
  salary      NUMBER(8,2);
BEGIN
  EXECUTE IMMEDIATE 'select first_name, salary from staffs where staff_id = :1'
    INTO first_name, salary
    USING IN staff_id;
  dbms_output.put_line(first_name || ' ' || salary);
END;
/

-- Invoke the stored procedure.
CALL dynamic_proc();

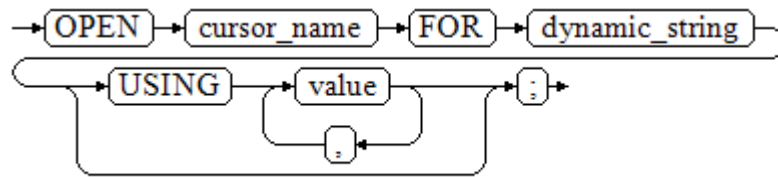
-- Delete the stored procedure.
DROP PROCEDURE dynamic_proc;
```

OPEN FOR

Dynamic query statements can be executed by using **OPEN FOR** to open dynamic cursors.

For details about the syntax, see [Figure 11-8](#).

Figure 11-8 open_for::=



Parameter description:

- **cursor_name**: specifies the name of the cursor to be opened.
- **dynamic_string**: specifies the dynamic query statement.
- **USING value**: applies when a placeholder exists in dynamic_string.

For use of cursors, see [DWS Stored Procedure Cursors](#).

Example

```
DECLARE
    name          VARCHAR2(20);
    phone_number  VARCHAR2(20);
    salary        NUMBER(8,2);
    sqlstr        VARCHAR2(1024);

    TYPE app_ref_cur_type IS REF CURSOR; -- Define the cursor type.
    my_cur app_ref_cur_type; -- Define the cursor variable.

BEGIN
    sqlstr := 'select first_name,phone_number,salary from staffs
               where section_id = :1';
    OPEN my_cur FOR sqlstr USING '30'; -- Open the cursor. using is optional.
    FETCH my_cur INTO name, phone_number, salary; -- Retrieve the data.
    WHILE my_cur%FOUND LOOP
        dbms_output.put_line(name||'#'||phone_number||'#'||salary);
        FETCH my_cur INTO name, phone_number, salary;
    END LOOP;
    CLOSE my_cur; -- Close the cursor.

END;
```

11.6.2 Executing Dynamic Non-query Statements

Syntax

Figure 11-9 shows the syntax diagram.

Figure 11-9 noselect::=

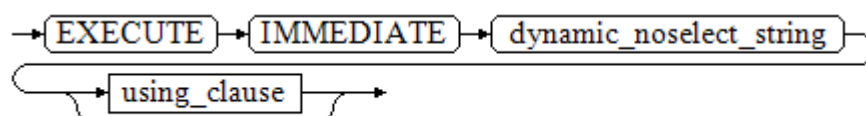
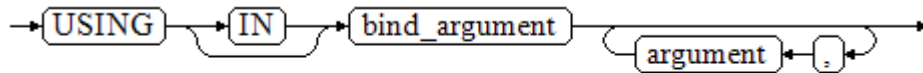


Figure 11-10 shows the syntax diagram for **using** clause.

Figure 11-10 using_clause-2

The above syntax diagram is explained as follows:

USING IN bind_argument is used to specify the variable that transfers values to dynamic SQL statements. It is used when a placeholder exists in **dynamic_noselect_string**. That is, a placeholder is replaced by the corresponding *bind_argument* when a dynamic SQL statement is executed. Note that *bind_argument* can only be a value, variable, or expression, and cannot be a database object such as a table name, column name, and data type. If a stored procedure needs to transfer database objects through *bind_argument* to construct dynamic SQL statements (generally, DDL statements), you are advised to use double vertical bars (||) to concatenate *dynamic_select_clause* with a database object. In addition, a dynamic PL/SQL block allows duplicate placeholders. That is, a placeholder can correspond to only one *bind_argument*.

Examples

```
-- Create a table:
CREATE TABLE sections_t1
(
    section      NUMBER(4) ,
    section_name VARCHAR2(30),
    manager_id   NUMBER(6),
    place_id     NUMBER(4)
)
DISTRIBUTE BY hash(manager_id);

--Declare a variable:
DECLARE
    section      NUMBER(4) := 280;
    section_name VARCHAR2(30) := 'Info support';
    manager_id   NUMBER(6) := 103;
    place_id     NUMBER(4) := 1400;
    new_colname  VARCHAR2(10) := 'sec_name';
BEGIN
-- Execute the query:
EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :4)'
    USING section, section_name, manager_id, place_id;
-- Execute the query (duplicate placeholders):
EXECUTE IMMEDIATE 'insert into sections_t1 values(:1, :2, :3, :1)'
    USING section, section_name, manager_id;
-- Run the ALTER statement. (You are advised to use double vertical bars (||) to concatenate the dynamic
DDL statement with a database object.)
EXECUTE IMMEDIATE 'alter table sections_t1 rename section_name to ' || new_colname;
END;
/

-- Query data:
SELECT * FROM sections_t1;

--Delete the table.
DROP TABLE sections_t1;
```

11.6.3 Dynamically Calling Stored Procedures

This section describes how to dynamically call store procedures. You must use anonymous statement blocks to package stored procedures or statement blocks

and append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

Syntax

Figure 11-11 shows the syntax diagram.

Figure 11-11 call_procedure::=

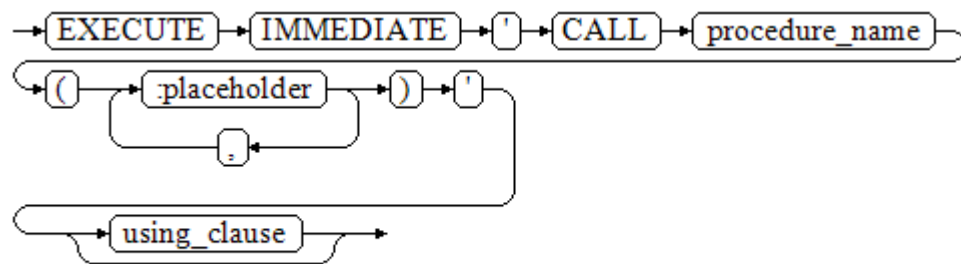
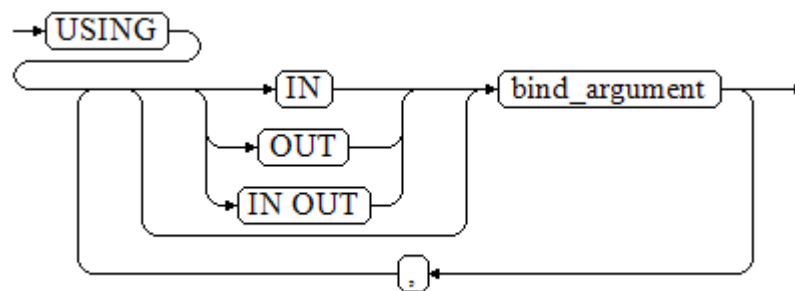


Figure 11-12 shows the syntax diagram for `using_clause`.

Figure 11-12 using_clause-3



The above syntax diagram is explained as follows:

- **CALL procedure_name:** calls the stored procedure.
- **[;placeholder1;placeholder2, ...]:** specifies the placeholder list of the stored procedure parameters. The number of placeholders is the same as the number of parameters. A placeholder name starts with a colon (:) or dollar sign (\$). The colon (:) can be followed by digits, characters, or character strings (excluding digits, characters, or character strings with quotation marks). The dollar sign (\$) can be followed only by digits. A placeholder can correspond to only one *bind_argument* in the **USING** clause.
- **USING [IN|OUT|IN OUT]bind_argument:** specifies where the variable passed to the stored procedure parameter value is stored. The modifiers in front of *bind_argument* and of the corresponding parameter are the same.

Examples

```
--Create the stored procedure proc_add.  
CREATE OR REPLACE PROCEDURE proc_add  
(
```

```
param1 in INTEGER,  
param2 out INTEGER,  
param3 in INTEGER  
)  
AS  
BEGIN  
    param2:= param1 + param3;  
END;  
/  
  
DECLARE  
    input1 INTEGER:=1;  
    input2 INTEGER:=2;  
    statement VARCHAR2(200);  
    param2 INTEGER;  
BEGIN  
    --Declare the call statement.  
    statement := 'call proc_add(:col_1, :col_2, :col_3)';(or statement := 'call proc_add($1, $2, $3)');  
    --Execute the statement.  
    EXECUTE IMMEDIATE statement  
        USING IN input1, OUT param2, IN input2;  
    dbms_output.put_line('result is: '||to_char(param2));  
END;  
/  
  
--Delete the stored procedure.  
DROP PROCEDURE proc_add;
```

11.6.4 Dynamically Calling Anonymous Blocks

This section describes how to execute anonymous blocks in dynamic statements. Append **IN** and **OUT** behind the **EXECUTE IMMEDIATE...USING** statement to input and output parameters.

Syntax

Figure 11-13 shows the syntax diagram.

Figure 11-13 call_anonymous_block::=

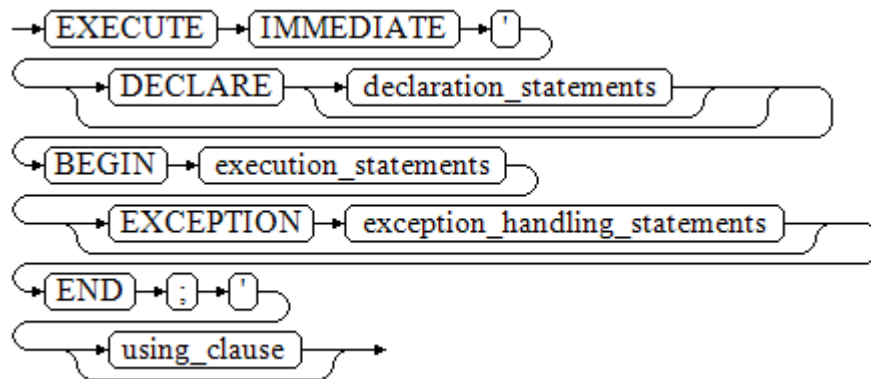
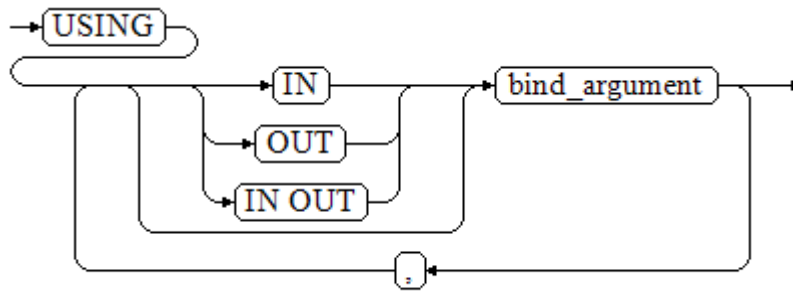


Figure 11-14 shows the syntax diagram for **using_clause**.

Figure 11-14 using_clause-4

The above syntax diagram is explained as follows:

- The execute part of an anonymous block starts with a **BEGIN** statement, has a break with an **END** statement, and ends with a semicolon (;).
- **USING [IN|OUT|IN OUT]bind_argument**: specifies where the variable passed to the stored procedure parameter value is stored. The modifiers in front of **bind_argument** and of the corresponding parameter are the same.
- The input and output parameters in the middle of an anonymous block are designated by placeholders. The numbers of the placeholders and the parameters are the same. The sequences of the parameters corresponding to the placeholders and the USING parameters are the same.
- Currently in DWS, when dynamic statements call anonymous blocks, placeholders cannot be used to pass input and output parameters in an **EXCEPTION** statement.

Example

```
--Create the stored procedure dynamic_proc.
CREATE OR REPLACE PROCEDURE dynamic_proc
AS
    staff_id    NUMBER(6) := 200;
    first_name  VARCHAR2(20);
    salary      NUMBER(8,2);
BEGIN
--Execute the anonymous block.
    EXECUTE IMMEDIATE 'begin select first_name, salary into :first_name, :salary from staffs where
staff_id= :dno; end;'
    USING OUT first_name, OUT salary, IN staff_id;
    dbms_output.put_line(first_name|| ' ' || salary);
END;
/

-- Invoke the stored procedure.
CALL dynamic_proc();

-- Delete the stored procedure.
DROP PROCEDURE dynamic_proc;
```

11.7 Control Statements of DWS Stored Procedure

11.7.1 RETURN Statements

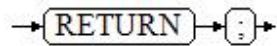
DWS provides two methods for returning data: **RETURN** (or **RETURN NEXT**) and **RETURN QUERY**. **RETURN NEXT** and **RETURN QUERY** are used only for functions and cannot be used for stored procedures.

RETURN

Syntax

Figure 11-15 shows the syntax of a return statement.

Figure 11-15 return_clause::=



The syntax is explained as follows:

This statement returns control from a stored procedure or function to a caller.

Examples

```
-- Create the stored procedure proc_staffs:
CREATE OR REPLACE PROCEDURE proc_staffs
(
  section    NUMBER(6),
  salary_sum out NUMBER(8,2),
  staffs_count out INTEGER
)
IS
BEGIN
  SELECT sum(salary), count(*) INTO salary_sum, staffs_count FROM staffs where section_id = section;
END;
/

-- Create the stored procedure proc_return:
CREATE OR REPLACE PROCEDURE proc_return
AS
  v_num NUMBER(8,2);
  v_sum INTEGER;
BEGIN
  proc_staffs(30, v_sum, v_num); --Call a statement.
  dbms_output.put_line(v_sum||'#'||v_num);
  RETURN; --Return a statement.
END;
/

-- Invoke a stored procedure proc_return:
CALL proc_return();

-- Delete a stored procedure:
DROP PROCEDURE proc_staffs;
DROP PROCEDURE proc_return;

--Create the function func_return.
CREATE OR REPLACE FUNCTION func_return returns void
language plpgsql
AS $$
DECLARE
  v_num INTEGER := 1;
BEGIN
  dbms_output.put_line(v_num);
  RETURN; --Return a statement.
```

```
END $$;

-- Invoke the function func_return.
CALL func_return();
1

-- Delete the function:
DROP FUNCTION func_return;
```

RETURN NEXT and RETURN QUERY

Syntax

When creating a function, specify **SETOF datatype** for the return values.

return_next_clause::=



return_query_clause::=



The syntax is explained as follows:

If a function needs to return a result set, use **RETURN NEXT** or **RETURN QUERY** to add results to the result set, and then continue to execute the next statement of the function. As the **RETURN NEXT** or **RETURN QUERY** statement is executed repeatedly, more and more results will be added to the result set. After the function is executed, all results are returned.

RETURN NEXT can be used for scalar and compound data types.

RETURN QUERY has a variant **RETURN QUERY EXECUTE**. You can add dynamic queries and add parameters to the queries by using **USING**.

Examples

```
CREATE TABLE t1(a int);
INSERT INTO t1 VALUES(1),(10);

--RETURN NEXT
CREATE OR REPLACE FUNCTION fun_for_return_next() RETURNS SETOF t1 AS $$
DECLARE
  r t1%ROWTYPE;
BEGIN
  FOR r IN select * from t1
  LOOP
    RETURN NEXT r;
  END LOOP;
  RETURN;
END;
$$ LANGUAGE PLPGSQL;
call fun_for_return_next();
a
---
1
10
(2 rows)

-- RETURN QUERY
```



```
CREATE OR REPLACE FUNCTION fun_for_return_query() RETURNS SETOF t1 AS $$
DECLARE
  r t1%ROWTYPE;
BEGIN
  RETURN QUERY select * from t1;
END;
$$
language plpgsql;
call fun_for_return_next();
a
---
1
10
(2 rows)
```

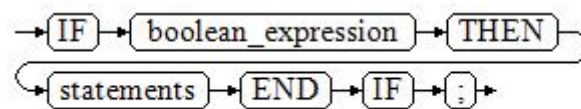
11.7.2 Conditional Statements

Conditional statements are used to decide whether given conditions are met. Operations are executed based on the decisions made.

DWS supports five usages of **IF**:

- **IF_THEN**

Figure 11-16 IF_THEN::=



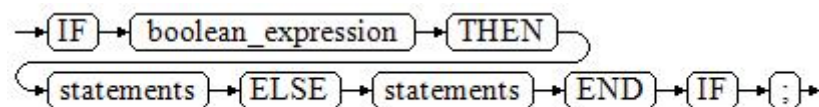
IF_THEN is the simplest form of **IF**. If the condition is true, statements are executed. If it is false, they are skipped.

Examples

```
IF v_user_id <> 0 THEN
  UPDATE users SET email = v_email WHERE user_id = v_user_id;
END IF;
```

- **IF_THEN_ELSE**

Figure 11-17 IF_THEN_ELSE::=



IF-THEN-ELSE statements add **ELSE** branches and can be executed if the condition is **false**.

Examples

```
IF parentid IS NULL OR parentid = ''
THEN
  RETURN;
ELSE
  hp_true_filename(parentid); -- Call the stored procedure.
END IF;
```

- **IF_THEN_ELSE IF**

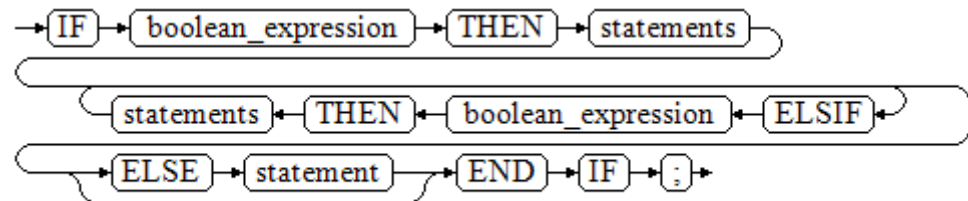
IF statements can be nested in the following way:

```
IF gender = 'm' THEN
    pretty_gender := 'man';
ELSE
    IF gender = 'f' THEN
        pretty_gender := 'woman';
    END IF;
END IF;
```

Actually, this is a way of an **IF** statement nesting in the **ELSE** part of another **IF** statement. Therefore, an **END IF** statement is required for each nesting **IF** statement and another **END IF** statement is required to end the parent **IF-ELSE** statement. To set multiple options, use the following form:

- IF_THEN_ELSEIF_ELSE

Figure 11-18 IF_THEN_ELSEIF_ELSE::=



Examples

```
IF number_tmp = 0 THEN
    result := 'zero';
ELSIF number_tmp > 0 THEN
    result := 'positive';
ELSIF number_tmp < 0 THEN
    result := 'negative';
ELSE
    result := 'NULL';
END IF;
```

- IF_THEN_ELSEIF_ELSE

ELSEIF is an alias of **ELSIF**.

Example

```
CREATE OR REPLACE PROCEDURE proc_control_structure(i in integer)
AS
BEGIN
    IF i > 0 THEN
        raise info 'i:% is greater than 0. ',i;
    ELSIF i < 0 THEN
        raise info 'i:% is smaller than 0. ',i;
    ELSE
        raise info 'i:% is equal to 0. ',i;
    END IF;
    RETURN;
END;
/

CALL proc_control_structure(3);

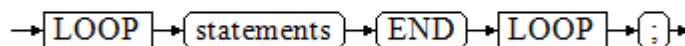
-- Delete the stored procedure.
DROP PROCEDURE proc_control_structure;
```

11.7.3 Loop Statements

Simple LOOP Statements

The syntax diagram is as follows.

Figure 11-19 loop::=



Examples

```
CREATE OR REPLACE PROCEDURE proc_loop(i in integer, count out integer)
AS
  BEGIN
    count:=0;
    LOOP
      IF count > i THEN
        raise info 'count is %.', count;
        EXIT;
      ELSE
        count:=count+1;
      END IF;
    END LOOP;
  END;
/

CALL proc_loop(10,5);
```

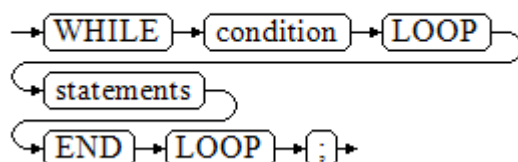
NOTICE

The loop must be exploited together with **EXIT**; otherwise, a dead loop occurs.

WHILE-LOOP Statements

The syntax diagram is as follows.

Figure 11-20 while_loop::=



If the conditional expression is true, a series of statements in the **WHILE** statement are repeatedly executed and the condition is decided each time the loop body is executed.

Examples

```
CREATE TABLE integertable(c1 integer) DISTRIBUTE BY hash(c1);
CREATE OR REPLACE PROCEDURE proc_while_loop(maxval in integer)
AS
  DECLARE
    i int :=1;
  BEGIN
    WHILE i < maxval LOOP
      INSERT INTO integertable VALUES(i);
      i:=i+1;
    END LOOP;
  END;
/

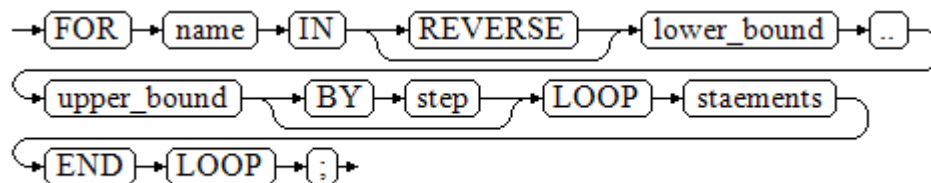
-- Invoke a function:
CALL proc_while_loop(10);

-- Delete the stored procedure and table:
DROP PROCEDURE proc_while_loop;
DROP TABLE integertable;
```

FOR_LOOP (*Integer variable*) Statement

The syntax diagram is as follows.

Figure 11-21 for_loop::=



NOTE

- The variable **name** is automatically defined as the **integer** type and exists only in this loop. The variable name falls between lower_bound and upper_bound.
- When the keyword **REVERSE** is used, the lower bound must be greater than or equal to the upper bound; otherwise, the loop body is not executed.

Examples

```
-- Loop from 0 to 5:
CREATE OR REPLACE PROCEDURE proc_for_loop()
AS
  BEGIN
    FOR I IN 0..5 LOOP
      DBMS_OUTPUT.PUT_LINE('It is '||to_char(I) || ' time;');
    END LOOP;
  END;
/

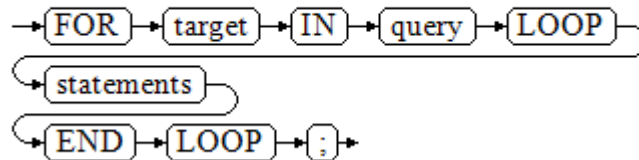
-- Invoke a function:
CALL proc_for_loop();

-- Delete the stored procedure:
DROP PROCEDURE proc_for_loop;
```

FOR_LOOP Query Statements

The syntax diagram is as follows.

Figure 11-22 for_loop_query::=



NOTE

The variable **target** is automatically defined, its type is the same as that in the **query** result, and it is valid only in this loop. The target value is the query result.

Examples

```
-- Display the query result from the loop:
CREATE OR REPLACE PROCEDURE proc_for_loop_query()
AS
    record VARCHAR2(50);
BEGIN
    FOR record IN SELECT spcname FROM pg_tablespace LOOP
        dbms_output.put_line(record);
    END LOOP;
END;
/

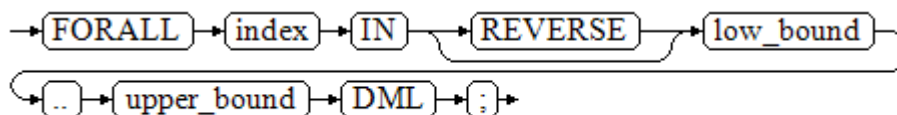
-- Invoke a function.
CALL proc_for_loop_query();

-- Delete the stored procedure.
DROP PROCEDURE proc_for_loop_query;
```

FORALL Batch Query Statements

The syntax diagram is as follows.

Figure 11-23 forall::=



NOTE

The variable **index** is automatically defined as the **integer** type and exists only in this loop. The index value falls between **low_bound** and **upper_bound**.

Examples

```
CREATE TABLE hdfs_t1 (
    title NUMBER(6),
```

```
did VARCHAR2(20),
data_period VARCHAR2(25),
kind VARCHAR2(25),
interval VARCHAR2(20),
time DATE,
isModified VARCHAR2(10)
)
DISTRIBUTE BY hash(did);

INSERT INTO hdfs_t1 VALUES( 8, 'Donald', 'OConnell', 'DOCONNEL', '650.507.9833', to_date('21-06-1999',
'dd-mm-yyyy'), 'SH_CLERK' );

CREATE OR REPLACE PROCEDURE proc_forall()
AS
BEGIN
    FORALL i IN 100..120
        insert into hdfs_t1(title) values(i);
END;
/

-- Invoke a function:
CALL proc_forall();

-- Query the invocation result of the stored procedure:
SELECT * FROM hdfs_t1 WHERE title BETWEEN 100 AND 120;

-- Delete the stored procedure and table:
DROP PROCEDURE proc_forall;
DROP TABLE hdfs_t1;
```

11.7.4 Branch Statements

Syntax

Figure 11-24 shows the syntax diagram.

Figure 11-24 case_when::=

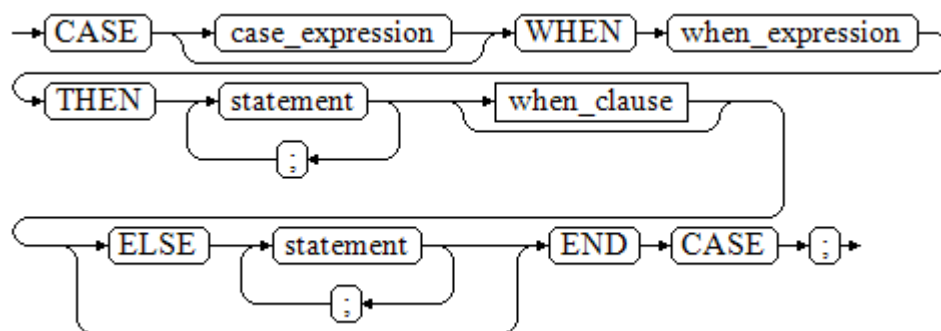
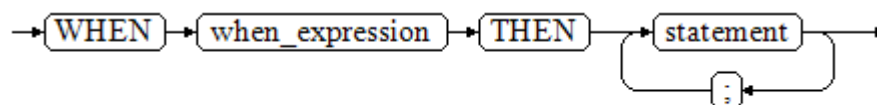


Figure 11-25 shows the syntax diagram for `when_clause`.

Figure 11-25 when_clause::=



Parameter description:

- **case_expression**: specifies the variable or expression.
- **when_expression**: specifies the constant or conditional expression.
- **statement**: specifies the statement to execute.

Examples

```
CREATE OR REPLACE PROCEDURE proc_case_branch(pi_result in integer, pi_return out integer)
AS
BEGIN
    CASE pi_result
        WHEN 1 THEN
            pi_return := 111;
        WHEN 2 THEN
            pi_return := 222;
        WHEN 3 THEN
            pi_return := 333;
        WHEN 6 THEN
            pi_return := 444;
        WHEN 7 THEN
            pi_return := 555;
        WHEN 8 THEN
            pi_return := 666;
        WHEN 9 THEN
            pi_return := 777;
        WHEN 10 THEN
            pi_return := 888;
        ELSE
            pi_return := 999;
        END CASE;
    raise info 'pi_return : %',pi_return ;
END;
/

CALL proc_case_branch(3,0);

--Delete the stored procedure.
DROP PROCEDURE proc_case_branch;
```

11.7.5 NULL Statements

In PL/SQL programs, a **NULL** statement can be used to indicate "do nothing", which is also known as an empty statement.

A NULL statement acts as a placeholder and can give meaning to certain statements, improving the readability of the program.

Syntax

The following shows example use of NULL statements.

```
DECLARE
...
BEGIN
...
    IF v_num IS NULL THEN
        NULL; --No data needs to be processed.
    END IF;
END;
/
```

11.7.6 Error Trapping Statements

By default, any error occurring in a PL/SQL function aborts execution of the function, and indeed of the surrounding transaction as well. You can trap errors

and restore from them by using a **BEGIN** block with an **EXCEPTION** clause. The syntax is an extension of the normal syntax for a **BEGIN** block:

```
[<<label>>]
[DECLARE
  declarations]
BEGIN
  statements
EXCEPTION
  WHEN condition [OR condition ...] THEN
    handler_statements
  [WHEN condition [OR condition ...] THEN
    handler_statements
  ...]
END;
```

If no error occurs, this form of block simply executes all the statements, and then control passes to the next statement after **END**. But if an error occurs inside the executed statement, the statement rolls back and goes to the **EXCEPTION** list to find the first condition that matches the error. If a match is found, the corresponding **handler_statements** are executed, and then control passes to the next statement after **END**. If no match is found, the error propagates out as though the **EXCEPTION** clause were not there at all:

The error can be caught by an enclosing block with **EXCEPTION**, or if there is none it aborts processing of the function.

The *condition* can be any of those shown in SQL standard error codes. The special condition name **OTHERS** matches every error type except **QUERY_CANCELED**.

If a new error occurs within the selected **handler_statements**, it cannot be caught by this **EXCEPTION** clause, but is propagated out. A surrounding **EXCEPTION** clause could catch it.

When an error is caught by an **EXCEPTION** clause, the local variables of the PL/SQL function remain as they were when the error occurred, but all changes to persistent database state within the block are rolled back.

Example:

```
CREATE TABLE mytab(id INT,firstname VARCHAR(20),lastname VARCHAR(20)) DISTRIBUTE BY hash(id);

INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');

CREATE FUNCTION fun_exp() RETURNS INT
AS $$
DECLARE
  x INT :=0;
  y INT;
BEGIN
  UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
  x := x + 1;
  y := x / 0;
EXCEPTION
  WHEN division_by_zero THEN
    RAISE NOTICE 'caught division_by_zero';
    RETURN x;
END;$$
LANGUAGE plpgsql;

CALL fun_exp();
NOTICE: caught division_by_zero
fun_exp
-----
      1
(1 row)
```



```
SELECT * FROM mytab;  
id | firstname | lastname  
-----  
  | Tom      | Jones  
(1 row)
```

```
DROP FUNCTION fun_exp();  
DROP TABLE mytab;
```

If a `division_by_zero` error occurs when a value is assigned to **y**, the error is caught by the **EXCEPTION** clause. The value returned in the **RETURN** statement will be the incremented value of **x**.

NOTE

A block containing an **EXCEPTION** clause is more expensive to enter and exit than a block without one. Therefore, do not use **EXCEPTION** without need.

In the following scenario, an exception cannot be caught, and the entire transaction rolls back. The threads of the nodes participating the stored procedure exit abnormally due to node failure and network fault, or the source data is inconsistent with that of the table structure of the target table during the `COPY FROM` operation.

Example: Exceptions with **UPDATE/INSERT**

Execute **UPDATE** or **INSERT** based on the exception handler.

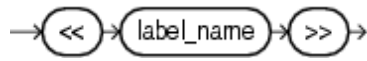
```
CREATE TABLE db (a INT, b TEXT);  
  
CREATE FUNCTION merge_db(key INT, data TEXT) RETURNS VOID AS  
$$  
BEGIN  
    LOOP  
  
    -- Try updating the key:  
    UPDATE db SET b = data WHERE a = key;  
    IF found THEN  
        RETURN;  
    END IF;  
  
    -- Not there, so try to insert the key. If someone else inserts the same key concurrently, we could get a  
    unique-key failure.  
    BEGIN  
        INSERT INTO db(a,b) VALUES (key, data);  
        RETURN;  
    EXCEPTION WHEN unique_violation THEN  
        -- Loop to try the UPDATE again:  
    END;  
    END LOOP;  
END;  
$$  
LANGUAGE plpgsql;  
  
SELECT merge_db(1, 'david');  
SELECT merge_db(1, 'dennis');  
  
-- Delete FUNCTION and TABLE:  
DROP FUNCTION merge_db;  
DROP TABLE db ;
```

11.7.7 GOTO Statements

The **GOTO** statement unconditionally transfers the control from the current statement to a labeled statement. The **GOTO** statement changes the execution logic. Therefore, use this statement only when necessary. Alternatively, you can use the **EXCEPTION** statement to handle issues in special scenarios. To run the **GOTO** statement, the labeled statement must be unique.

Syntax

label declaration ::=



goto statement ::=



Examples

```
CREATE OR REPLACE PROCEDURE GOTO_test()
AS
DECLARE
    v1 int;
BEGIN
    v1 := 0;
    LOOP
        EXIT WHEN v1 > 100;
        v1 := v1 + 2;
        if v1 > 25 THEN
            GOTO pos1;
        END IF;
    END LOOP;
    <<pos1>>
    v1 := v1 + 10;
    raise info 'v1 is %.', v1;
END;
/

call GOTO_test();
DROP PROCEDURE GOTO_test();
```

Constraints

The **GOTO** statement has the following constraints:

- The **GOTO** statement does not allow multiple labeled statements even if they are in different blocks.

```
BEGIN
    GOTO pos1;
    <<pos1>>
    SELECT * FROM ...
    <<pos1>>
    UPDATE t1 SET ...
END;
```

- The **GOTO** statement cannot transfer control to the **IF**, **CASE**, or **LOOP** statement.

```
BEGIN
    GOTO pos1;
    IF valid THEN
        <<pos1>>
        SELECT * FROM ...
    END IF;
END;
```

- The **GOTO** statement cannot transfer control from one **IF** clause to another, or from one **WHEN** clause in the **CASE** statement to another.

```
BEGIN
    IF valid THEN
        GOTO pos1;
        SELECT * FROM ...
    END IF;
```

```
ELSE
  <<pos1>>
  UPDATE t1 SET ...
END IF;
END;
```

- The **GOTO** statement cannot transfer control from an outer block to an inner **BEGIN-END** block.

```
BEGIN
  GOTO pos1;
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
END;
END;
```

- The **GOTO** statement cannot transfer control from an **EXCEPTION** block to the current **BEGIN-END** block but can transfer to an outer **BEGIN-END** block.

```
BEGIN
  <<pos1>>
  UPDATE t1 SET ...
  EXCEPTION
    WHEN condition THEN
      GOTO pos1;
END;
```

- If the labeled statement in the **GOTO** statement does not exist, you need to add the **NULL** statement.

```
DECLARE
  done BOOLEAN;
BEGIN
  FOR i IN 1..50 LOOP
    IF done THEN
      GOTO end_loop;
    END IF;
    <<end_loop>> -- not allowed unless an executable statement follows
    NULL; -- add NULL statement to avoid error
  END LOOP; -- raises an error without the previous NULL
END;
/
```

11.8 Other Statements in DWS Stored Procedures

Lock Operations

DWS provides multiple lock modes to control concurrent accesses to table data. These modes are used when Multi-Version Concurrency Control (MVCC) cannot give expected behaviors. Alike, most DWS commands automatically apply appropriate locks to ensure that called tables are not deleted or modified in an incompatible manner during command execution. For example, when concurrent operations exist, **ALTER TABLE** cannot be executed on the same table.

Cursor Operations

DWS provides cursors as a data buffer for users to store execution results of SQL statements. Each cursor region has a name. Users can use SQL statements to obtain records one by one from cursors and grant them to master variables, then being processed further by host languages.

Cursor operations include cursor definition, open, fetch, and close operations.

For the complete example of cursor operations, see [Explicit Cursor](#).

11.9 DWS Stored Procedure Cursors

11.9.1 Overview

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

NOTICE

If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor is not available.

Cursors are classified into explicit cursors and implicit cursors. [Table 11-2](#) shows the usage conditions of explicit and implicit cursors for different SQL statements.

Table 11-2 Cursor usage conditions

SQL Statement	Cursor
Non-query statements	Implicit
Query statements with single-line results	Implicit or explicit
Query statements with multi-line results	Explicit

11.9.2 Explicit Cursor

An explicit cursor is used to process query statements, particularly when the query results contain multiple records.

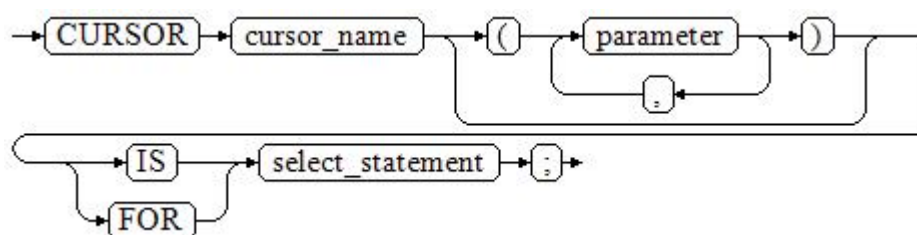
Procedure

An explicit cursor performs the following six PL/SQL steps to process query statements:

- Step 1 Define a static cursor:** Define a cursor name and its corresponding **SELECT** statement.

[Figure 11-26](#) shows the syntax diagram for defining a static cursor.

Figure 11-26 static_cursor_define::=



Parameter description:

- **cursor_name**: defines a cursor name.
- **parameter**: specifies cursor parameters. Only input parameters are allowed in the following format:
parameter_name datatype
- **select_statement**: specifies a query statement.

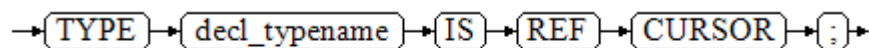
 NOTE

The system automatically determines whether the cursor can be used for backward fetches based on the execution plan.

Define a dynamic cursor: Define a **ref** cursor, which means that the cursor can be opened dynamically by a set of static SQL statements. Define the type of the **ref** cursor first and then the cursor variable of this cursor type. Dynamically bind a **SELECT** statement through **OPEN FOR** when the cursor is opened.

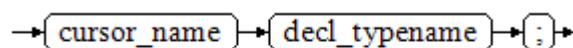
Figure 11-27 and **Figure 11-28** show the syntax diagrams for defining a dynamic cursor.

Figure 11-27 cursor_type_name::=



DWS supports the dynamic cursor type **sys_refcursor**. A function or stored procedure can use the **sys_refcursor** parameter to pass on or pass out the cursor result set. A function can return **sys_refcursor** to return the cursor result set.

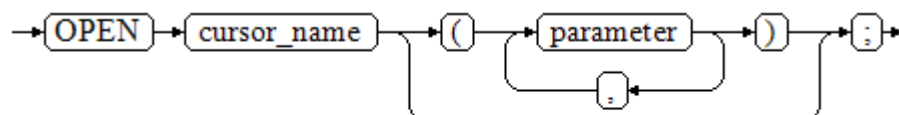
Figure 11-28 dynamic_cursor_define::=



Step 2 Open the static cursor: Execute the **SELECT** statement corresponding to the cursor. The query result is placed in the work area and the pointer directs to the head of the work area to identify the cursor result set. If the cursor query statement contains the **FOR UPDATE** option, the **OPEN** statement locks the data row corresponding to the cursor result set in the database table.

Figure 11-29 shows the syntax diagram for opening a static cursor.

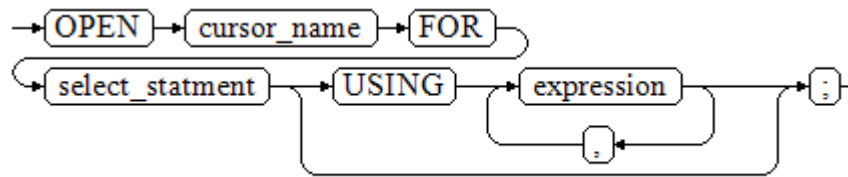
Figure 11-29 open_static_cursor::=



Open the dynamic cursor: Use the **OPEN FOR** statement to open the dynamic cursor and the SQL statement is dynamically bound.

Figure 11-30 shows the syntax diagram for opening a dynamic cursor.

Figure 11-30 open_dynamic_cursor::=

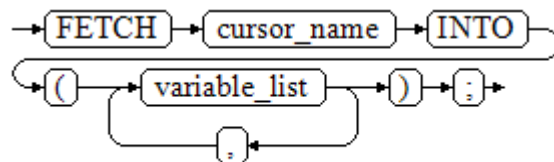


A PL/SQL program cannot use the **OPEN** statement to repeatedly open a cursor.

Step 3 Fetch cursor data: Retrieve data rows in the result set and place them in specified output variables.

Figure 11-31 shows the syntax diagram for fetching cursor data.

Figure 11-31 fetch_cursor::=



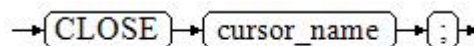
Step 4 Process the record.

Step 5 Continue to process until the active set has no record.

Step 6 Close the cursor: When fetching and finishing the data in the cursor result set, close the cursor immediately to release system resources used by the cursor and invalidate the work area of the cursor so that the **FETCH** statement cannot be used to fetch data any more. A closed cursor can be reopened using the **OPEN** statement.

Figure 11-32 shows the syntax diagram for closing a cursor.

Figure 11-32 close_cursor::=



----End

Attributes

Cursor attributes are used to control program procedures or learn about program status. When a DML statement is executed, the PL/SQL opens a built-in cursor and processes its result. A cursor is a memory segment for maintaining query results. It is opened when a DML statement is executed and closed when the execution is finished. An explicit cursor has the following attributes:

- **%FOUND:** Boolean attribute, which returns **TRUE** if the last fetch returns a row.

- **%NOTFOUND**: Boolean attribute, which works opposite to the **%FOUND** attribute.
- **%ISOPEN**: Boolean attribute, which returns **TRUE** if the cursor has been opened.
- **%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.

Examples

```
-- Specify the method for passing cursor parameters:
CREATE OR REPLACE PROCEDURE cursor_proc1()
AS
DECLARE
    DEPT_NAME VARCHAR(100);
    DEPT_LOC NUMBER(4);
    -- Define a cursor:
    CURSOR C1 IS
        SELECT section_name, place_id FROM sections WHERE section_id <= 50;
    CURSOR C2(sect_id INTEGER) IS
        SELECT section_name, place_id FROM sections WHERE section_id <= sect_id;
    TYPE CURSOR_TYPE IS REF CURSOR;
    C3 CURSOR_TYPE;
    SQL_STR VARCHAR(100);
BEGIN
    OPEN C1;-- Open the cursor:
    LOOP
        -- Fetch data from the cursor:
        FETCH C1 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C1;-- Close the cursor.

    OPEN C2(10);
    LOOP
        FETCH C2 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C2%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C2;

    SQL_STR := 'SELECT section_name, place_id FROM sections WHERE section_id <= :DEPT_NO;';
    OPEN C3 FOR SQL_STR USING 50;
    LOOP
        FETCH C3 INTO DEPT_NAME, DEPT_LOC;
        EXIT WHEN C3%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(DEPT_NAME||'---'||DEPT_LOC);
    END LOOP;
    CLOSE C3;
END;
/

CALL cursor_proc1();

DROP PROCEDURE cursor_proc1;
-- Increase the salary of employees whose salary is lower than CNY3000 by CNY500:
CREATE TABLE staffs_t1 AS TABLE staffs;

CREATE OR REPLACE PROCEDURE cursor_proc2()
AS
DECLARE
    V_EMPNO NUMBER(6);
    V_SAL NUMBER(8,2);
    CURSOR C IS SELECT staff_id, salary FROM staffs_t1;
BEGIN
    OPEN C;
```

```
LOOP
  FETCH C INTO V_EMPNO, V_SAL;
  EXIT WHEN C%NOTFOUND;
  IF V_SAL<=3000 THEN
    UPDATE staffs_t1 SET salary =salary + 500 WHERE staff_id = V_EMPNO;
  END IF;
END LOOP;
CLOSE C;
END;
/

CALL cursor_proc2();

-- Drop the stored procedure:
DROP PROCEDURE cursor_proc2;
DROP TABLE staffs_t1;
-- Use function parameters of the SYS_REFCURSOR type:
CREATE OR REPLACE PROCEDURE proc_sys_ref(O OUT SYS_REFCURSOR)
IS
C1 SYS_REFCURSOR;
BEGIN
  OPEN C1 FOR SELECT section_ID FROM sections ORDER BY section_ID;
  O := C1;
END;
/

DECLARE
C1 SYS_REFCURSOR;
TEMP NUMBER(4);
BEGIN
  proc_sys_ref(C1);
  LOOP
    FETCH C1 INTO TEMP;
    DBMS_OUTPUT.PUT_LINE(C1%ROWCOUNT);
    EXIT WHEN C1%NOTFOUND;
  END LOOP;
END;
/

-- Drop the stored procedure:
DROP PROCEDURE proc_sys_ref;
```

11.9.3 Implicit Cursor

The system automatically sets implicit cursors for non-query statements, such as **ALTER** and **DROP**, and creates work areas for these statements. These implicit cursors are named SQL, which is defined by the system.

Overview

Implicit cursor operations, such as definition, opening, value-grant, and closing, are automatically performed by the system. Users can use only the attributes of implicit cursors to complete operations. The data stored in the work area of an implicit cursor is the latest SQL statement, and is not related to the user-defined explicit cursors.

Format call: SQL%

NOTE

INSERT, **UPDATE**, **DROP**, and **SELECT** statements do not require defined cursors.

Attributes

An implicit cursor has the following attributes:

- **SQL%FOUND**: Boolean attribute, which returns **TRUE** if the last fetch returns a row.
- **SQL%NOTFOUND**: Boolean attribute, which works opposite to the **SQL%FOUND** attribute.
- **SQL%ROWCOUNT**: numeric attribute, which returns the number of records fetched from the cursor.
- **SQL%ISOPEN**: Boolean attribute, whose value is always **FALSE**. Close implicit cursors immediately after an SQL statement is executed.

Examples

```
-- Delete all employees in a department from the EMP table. If the department has no employees, delete
the department from the DEPT table.
CREATE TABLE staffs_t1 AS TABLE staffs;
CREATE TABLE sections_t1 AS TABLE sections;

CREATE OR REPLACE PROCEDURE proc_cursor3()
AS
  DECLARE
    V_DEPTNO NUMBER(4) := 100;
  BEGIN
    DELETE FROM staffs WHERE section_ID = V_DEPTNO;
    -- Proceed based on cursor status:
    IF SQL%NOTFOUND THEN
      DELETE FROM sections_t1 WHERE section_ID = V_DEPTNO;
    END IF;
  END;
/

CALL proc_cursor3();

-- Drop the stored procedure and the temporary table:
DROP PROCEDURE proc_cursor3;
DROP TABLE staffs_t1;
DROP TABLE sections_t1;
```

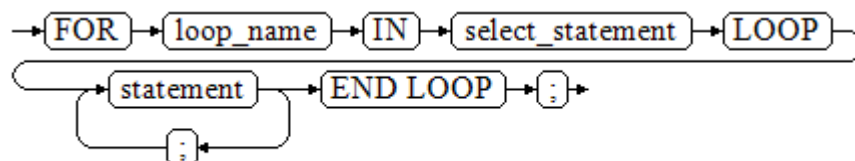
11.9.4 Cursor Loop

The use of cursors in **WHILE** and **LOOP** statements is called a cursor loop. Generally, **OPEN**, **FETCH**, and **CLOSE** statements are needed in cursor loop. The following describes a loop that is applicable to a static cursor loop without executing the four steps of a static cursor.

Syntax

Figure 11-33 shows the syntax diagram for the **FOR AS** loop.

Figure 11-33 FOR_AS_loop::=



Precautions

- The **UPDATE** operation for the queried table is not allowed in the loop statement.
- The variable *loop_name* is automatically defined and is valid only in this loop. The type and value of *loop_name* are the same as those of the query result of *select_statement*.
- The **%FOUND**, **%NOTFOUND**, and **%ROWCOUNT** attributes access the same internal variable in DWS. Transactions and anonymous blocks cannot be accessed by multiple cursors at the same time.

Examples

```
BEGIN
FOR ROW_TRANS IN
  SELECT first_name FROM staffs
  LOOP
    DBMS_OUTPUT.PUT_LINE (ROW_TRANS.first_name );
  END LOOP;
END;
/

-- Create a table:
CREATE TABLE integerTable1( A INTEGER) DISTRIBUTE BY hash(A);
CREATE TABLE integerTable2( B INTEGER) DISTRIBUTE BY hash(B);
INSERT INTO integerTable2 VALUES(2);

-- Multiple cursors share the parameters of cursor attributes:
DECLARE
  CURSOR C1 IS SELECT A FROM integerTable1;--Declare the cursor.
  CURSOR C2 IS SELECT B FROM integerTable2;
  PI_A INTEGER;
  PI_B INTEGER;
BEGIN
  OPEN C1;-- Open the cursor.
  OPEN C2;
  FETCH C1 INTO PI_A; ---- The value of C1%FOUND and C2%FOUND is FALSE.
  FETCH C2 INTO PI_B; ---- The value of C1%FOUND and C2%FOUND is TRUE.
-- Determine the cursor status:
  IF C1%FOUND THEN
    IF C2%FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Dual cursor share parameter.');
```

11.10 Advanced Packages of DWS Stored Procedures

11.10.1 DBMS_LOB

Related Interfaces

[Table 11-3](#) provides all interfaces supported by the **DBMS_LOB** package.

Table 11-3 DBMS_LOB

API	Description
DBMS_LOB.GETLENGTH	Obtains and returns the specified length of a LOB object.
DBMS_LOB.OPEN	Opens a LOB and returns a LOB descriptor.
DBMS_LOB.READ	Loads a part of LOB contents to BUFFER area according to the specified length and initial position offset.
DBMS_LOB.WRITE	Copies contents in BUFFER area to LOB according to the specified length and initial position offset.
DBMS_LOB.WRITEAPPEND	Copies contents in BUFFER area to the end part of LOB according to the specified length.
DBMS_LOB.COPY	Copies contents in BLOB to another BLOB according to the specified length and initial position offset.
DBMS_LOB.ERASE	Deletes contents in BLOB according to the specified length and initial position offset.
DBMS_LOB.CLOSE	Closes a LOB descriptor.
DBMS_LOB.INSTR	Returns the position of the Nth occurrence of a character string in LOB.
DBMS_LOB.COMPARE	Compares two LOBs or a certain part of two LOBs.
DBMS_LOB.SUBSTR	Reads the substring of a LOB and returns the number of read bytes or the number of characters.
DBMS_LOB.TRIM	Truncates the LOB of a specified length. After the execution is complete, the length of the LOB is set to the length specified by the newlen parameter.
DBMS_LOB.CREATETEMPORARY	Creates a temporary BLOB or CLOB.
DBMS_LOB.APPEND	Adds the content of a LOB to another LOB.

- **DBMS_LOB.GETLENGTH**

Specifies the length of a LOB type object obtained and returned by the stored procedure **GETLENGTH**.

The function prototype of **DBMS_LOB.GETLENGTH** is:

```
DBMS_LOB.GETLENGTH (  
lob_loc IN BLOB)  
RETURN INTEGER;
```

```
DBMS_LOB.GETLENGTH (  
lob_loc IN CLOB)  
RETURN INTEGER;
```

Table 11-4 DBMS_LOB.GETLENGTH interface parameters

Parameter	Description
lob_loc	LOB type object whose length is to be obtained

- DBMS_LOB.OPEN

A stored procedure opens a LOB and returns a LOB descriptor. This process is used only for compatibility.

The function prototype of **DBMS_LOB.OPEN** is:

```
DBMS_LOB.LOB (  
lob_loc INOUT BLOB,  
open_mode IN BINARY_INTEGER);
```

```
DBMS_LOB.LOB (  
lob_loc INOUT CLOB,  
open_mode IN BINARY_INTEGER);
```

Table 11-5 DBMS_LOB.OPEN interface parameters

Parameter	Description
lob_loc	BLOB or CLOB descriptor that is opened
open_mode IN BINARY_INTEGER	Open mode (currently, DBMS_LOB.LOB_READWRITE is supported)

- DBMS_LOB.READ

The stored procedure **READ** loads a part of LOB contents to BUFFER according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.READ** is:

```
DBMS_LOB.READ (  
lob_loc IN BLOB,  
amount IN INTEGER,  
offset IN INTEGER,  
buffer OUT RAW);
```

```
DBMS_LOB.READ (  
lob_loc IN CLOB,  
amount IN OUT INTEGER,  
offset IN INTEGER,  
buffer OUT VARCHAR2);
```

Table 11-6 DBMS_LOB.READ interface parameters

Parameter	Description
lob_loc	LOB type object to be loaded
amount	Load data length NOTE If the read length is negative, the error message "ERROR: argument 2 is null, invalid, or out of range." is displayed.

Parameter	Description
offset	Indicates where to start reading the LOB contents, that is, the offset bytes to initial position of LOB contents.
buffer	Target buffer to store the loaded LOB contents

- **DBMS_LOB.WRITE**

The stored procedure **WRITE** copies contents in **BUFFER** to LOB variables according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.WRITE** is:

```
DBMS_LOB.WRITE (  
lob_loc IN OUT BLOB,  
amount IN INTEGER,  
offset IN INTEGER,  
buffer IN RAW);  
  
DBMS_LOB.WRITE (  
lob_loc IN OUT CLOB,  
amount IN INTEGER,  
offset IN INTEGER,  
buffer IN VARCHAR2);
```

Table 11-7 DBMS_LOB.WRITE interface parameters

Parameter	Description
lob_loc	LOB type object to be written
amount	Write data length NOTE If the write data is shorter than 1 or longer than the contents to be written, an error is reported.
offset	Indicates where to start writing the LOB contents, that is, the offset bytes to initial position of LOB contents. NOTE If the offset is shorter than 1 or longer than the maximum length of LOB type contents, an error is reported.
buffer	Content to be written

- **DBMS_LOB.WRITEAPPEND**

The stored procedure **WRITEAPPEND** copies contents in **BUFFER** to the end part of LOB according to the specified length.

The function prototype of **DBMS_LOB.WRITEAPPEND** is:

```
DBMS_LOB.WRITEAPPEND (  
lob_loc IN OUT BLOB,  
amount IN INTEGER,  
buffer IN RAW);  
  
DBMS_LOB.WRITEAPPEND (  
lob_loc IN OUT CLOB,  
amount IN INTEGER,  
buffer IN VARCHAR2);
```

Table 11-8 DBMS_LOB.WRITEAPPEND interface parameters

Parameter	Description
lob_loc	LOB type object to be written
amount	Write data length NOTE If the write data is shorter than 1 or longer than the contents to be written, an error is reported.
buffer	Content to be written

- **DBMS_LOB.COPY**

The stored procedure **COPY** copies contents in BLOB to another BLOB according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.COPY** is:

```
DBMS_LOB.COPY (  
dest_lob IN OUT BLOB,  
src_lob IN BLOB,  
amount IN INTEGER,  
dest_offset IN INTEGER DEFAULT 1,  
src_offset IN INTEGER DEFAULT 1);
```

Table 11-9 DBMS_LOB.COPY interface parameters

Parameter	Description
dest_lob	BLOB type object to be pasted
src_lob	BLOB type object to be copied
amount	Replication length. NOTE If the copied data is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
dest_offset	Indicates where to start pasting the BLOB contents, that is, the offset bytes to initial position of BLOB contents. NOTE If the offset is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
src_offset	Indicates where to start copying the BLOB contents, that is, the offset bytes to initial position of BLOB contents. NOTE If the offset is shorter than 1 or longer than the length of source BLOB, an error is reported.

- **DBMS_LOB.ERASE**

The stored procedure **ERASE** deletes contents in BLOB according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.ERASE** is:

```
DBMS_LOB.ERASE (  
lob_loc IN OUT BLOB,
```

```
amount      IN OUT  INTEGER,  
offset      IN     INTEGER DEFAULT 1);
```

Table 11-10 DBMS_LOB.ERASE interface parameters

Parameter	Description
lob_loc	BLOB type object whose contents are to be deleted
amount	Length of contents to be deleted NOTE If the deleted data is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.
offset	Indicates where to start deleting the BLOB contents, that is, the offset bytes to initial position of BLOB contents. NOTE If the offset is shorter than 1 or longer than the maximum length of BLOB type contents, an error is reported.

- DBMS_LOB.CLOSE

The procedure **CLOSE** disables the enabled contents of LOB according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.CLOSE** is:

```
DBMS_LOB.CLOSE(  
src_lob      IN          BLOB);  
  
DBMS_LOB.CLOSE (  
src_lob      IN          CLOB);
```

Table 11-11 DBMS_LOB.CLOSE interface parameters

Parameter	Description
src_loc	LOB type object to be disabled

- DBMS_LOB.INSTR

This function returns the Nth occurrence position in LOB. If invalid values are entered, **NULL** is returned. The invalid values include offset < 1 or offset > LOBMAXSIZE, nth < 1, and nth > LOBMAXSIZE.

The function prototype of **DBMS_LOB.INSTR** is:

```
DBMS_LOB.INSTR (  
lob_loc      IN  BLOB,  
pattern      IN  RAW,  
offset       IN  INTEGER := 1,  
nth          IN  INTEGER := 1)  
RETURN INTEGER;  
  
DBMS_LOB.INSTR (  
lob_loc      IN  CLOB,  
pattern      IN  VARCHAR2 ,  
offset       IN  INTEGER := 1,  
nth          IN  INTEGER := 1)  
RETURN INTEGER;
```

Table 11-12 DBMS_LOB.INSTR interface parameters

Parameter	Description
lob_loc	LOB descriptor to be searched for
pattern	Matched pattern. It is RAW for BLOB and TEXT for CLOB.
offset	For BLOB, the absolute offset is in the unit of byte. For CLOB, the offset is in the unit of character. The matching start position is 1.
nth	Number of pattern matching times. The minimum value is 1.

- DBMS_LOB.COMPARE

This function compares two LOBs or a certain part of two LOBs.

- If the two parts are equal, **0** is returned. Otherwise, a non-zero value is returned.
- If the first CLOB is smaller than the second, **-1** is returned. If the first CLOB is larger than the second, **1** is returned.
- If any of the **amount**, **offset_1**, and **offset_2** parameters is invalid, **NULL** is returned. The valid offset range is 1 to LOBMAXSIZE.

The function prototype of **DBMS_LOB.READ** is:

```
DBMS_LOB.COMPARE (  
lob_1   IN   BLOB,  
lob_2   IN   BLOB,  
amount  IN   INTEGER := DBMS_LOB.LOBMAXSIZE,  
offset_1 IN   INTEGER := 1,  
offset_2 IN   INTEGER := 1)  
RETURN INTEGER;  
  
DBMS_LOB.COMPARE (  
lob_1   IN   CLOB,  
lob_2   IN   CLOB,  
amount  IN   INTEGER := DBMS_LOB.LOBMAXSIZE,  
offset_1 IN   INTEGER := 1,  
offset_2 IN   INTEGER := 1)  
RETURN INTEGER;
```

Table 11-13 DBMS_LOB.COMPARE interface parameters

Parameter	Description
lob_1	First LOB descriptor to be compared
lob_2	Second LOB descriptor to be compared
amount	Number of characters or bytes to be compared. The maximum value is DBMS_LOB.LOBMAXSIZE.
offset_1	Offset of the first LOB descriptor. The initial position is 1.
offset_2	Offset of the second LOB descriptor. The initial position is 1.

- DBMS_LOB.SUBSTR

This function reads the substring of a LOB and returns the number of read bytes or the number of characters. If amount > 1, amount < 32767, offset < 1, or offset > LOBMAXSIZE, **NULL** is returned.

The function prototype of **DBMS_LOB.SUBSTR** is:

```
DBMS_LOB.SUBSTR (  
lob_loc  IN   BLOB,  
amount   IN   INTEGER := 32767,  
offset   IN   INTEGER := 1)  
RETURN RAW;
```

```
DBMS_LOB.SUBSTR (  
lob_loc  IN   CLOB,  
amount   IN   INTEGER := 32767,  
offset   IN   INTEGER := 1)  
RETURN VARCHAR2;
```

Table 11-14 DBMS_LOB.SUBSTR interface parameters

Parameter	Description
lob_loc	LOB descriptor of the substring to be read. For BLOB, the return value is the number of read bytes. For CLOB, the return value is the number of characters.
offset	Number of bytes or characters to be read.
buffer	Number of characters or bytes offset from the start position.

- DBMS_LOB.TRIM

This stored procedure truncates the LOB of a specified length. After this stored procedure is executed, the length of the LOB is set to the length specified by the **newlen** parameter. If an empty LOB is truncated, no execution result is displayed. If the specified length is longer than the length of LOB, an exception occurs.

The function prototype of **DBMS_LOB.TRIM** is:

```
DBMS_LOB.TRIM (  
lob_loc  IN OUT   BLOB,  
newlen   IN       INTEGER);
```

```
DBMS_LOB.TRIM (  
lob_loc  IN       OUT CLOB,  
newlen   IN       INTEGER);
```

Table 11-15 DBMS_LOB.TRIM interface parameters

Parameter	Description
lob_loc	BLOB type object to be read
newlen	After truncation, the new LOB length for BLOB is in the unit of byte and that for CLOB is in the unit of character.

- DBMS_LOB.CREATETEMPORARY

This stored procedure creates a temporary BLOB or CLOB and is used only for syntax compatibility.

The function prototype of **DBMS_LOB.CREATETEMPORARY** is:

```
DBMS_LOB.CREATETEMPORARY (  
lob_loc IN OUT BLOB,  
cache IN BOOLEAN,  
dur IN INTEGER);
```

```
DBMS_LOB.CREATETEMPORARY (  
lob_loc IN OUT CLOB,  
cache IN BOOLEAN,  
dur IN INTEGER);
```

Table 11-16 DBMS_LOB.CREATETEMPORARY interface parameters

Parameter	Description
lob_loc	LOB descriptor
cache	This parameter is used only for syntax compatibility.
dur	This parameter is used only for syntax compatibility.

- DBMS_LOB.APPEND

The stored procedure **READ** loads a part of BLOB contents to BUFFER according to the specified length and initial position offset.

The function prototype of **DBMS_LOB.APPEND** is:

```
DBMS_LOB.APPEND (  
dest_lob IN OUT BLOB,  
src_lob IN BLOB);
```

```
DBMS_LOB.APPEND (  
dest_lob IN OUT CLOB,  
src_lob IN CLOB);
```

Table 11-17 DBMS_LOB.APPEND interface parameters

Parameter	Description
dest_lob	LOB descriptor to be written
src_lob	LOB descriptor to be read

Examples

```
-- Obtain the length of the character string.  
SELECT DBMS_LOB.GETLENGTH('12345678');  
  
DECLARE  
myraw RAW(100);  
amount INTEGER :=2;  
buffer INTEGER :=1;  
begin  
DBMS_LOB.READ('123456789012345',amount,buffer,myraw);  
dbms_output.put_line(myraw);  
end;
```

```
/
CREATE TABLE blob_Table (t1 blob) DISTRIBUTE BY REPLICATION;
CREATE TABLE blob_Table_bak (t2 blob) DISTRIBUTE BY REPLICATION;
INSERT INTO blob_Table VALUES('abcdef');
INSERT INTO blob_Table_bak VALUES('22222');

DECLARE
str varchar2(100) := 'abcdef';
source raw(100);
dest blob;
copyto blob;
amount int;
PSV_SQL varchar2(100);
PSV_SQL1 varchar2(100);
a int :=1;
len int;
BEGIN
source := utl_raw.cast_to_raw(str);
amount := utl_raw.length(source);

PSV_SQL := 'select * from blob_Table for update';
PSV_SQL1 := 'select * from blob_Table_bak for update';

EXECUTE IMMEDIATE PSV_SQL into dest;
EXECUTE IMMEDIATE PSV_SQL1 into copyto;

DBMS_LOB.WRITE(dest, amount, 1, source);
DBMS_LOB.WRITEAPPEND(dest, amount, source);

DBMS_LOB.ERASE(dest, a, 1);
DBMS_OUTPUT.PUT_LINE(a);
DBMS_LOB.COPY(copyto, dest, amount, 10, 1);
DBMS_LOB.CLOSE(dest);
RETURN;
END;
/

--Delete the table.
DROP TABLE blob_Table;
DROP TABLE blob_Table_bak;
```

11.10.2 DBMS_RANDOM

Related Interfaces

Table 11-18 provides all interfaces supported by the **DBMS_RANDOM** package.

Table 11-18 DBMS_RANDOM interface parameters

API	Description
DBMS_RANDOM.SEED	Sets a seed for a random number.
DBMS_RANDOM.VALUE	Generates a random number between a specified low and a specified high.

- **DBMS_RANDOM.SEED**
The stored procedure SEED is used to set a seed for a random number. The DBMS_RANDOM.SEED function prototype is:

```
DBMS_RANDOM.SEED (seed IN INTEGER);
```

Table 11-19 DBMS_RANDOM.SEED interface parameters

Parameter	Description
seed	Generates a seed for a random number.

- DBMS_RANDOM.VALUE

The stored procedure VALUE generates a random number between a specified low and a specified high. The DBMS_RANDOM.VALUE function prototype is:

```
DBMS_RANDOM.VALUE(  
low IN NUMBER,  
high IN NUMBER)  
RETURN NUMBER;
```

Table 11-20 DBMS_RANDOM.VALUE interface parameters

Parameter	Description
low	Sets the low bound for a random number. The generated random number is greater than or equal to the low.
high	Sets the high bound for a random number. The generated random number is less than the high.

 NOTE

The only requirement is that the parameter type is **NUMERIC** regardless of the right and left bound values.

Example

Generate a random number between 0 and 1:

```
SELECT DBMS_RANDOM.VALUE(0,1);
```

Generate a random integer ranging from 0 to 100. The random integer is greater than or equal to the specified value of low and less than the specified value of high.

```
SELECT TRUNC(DBMS_RANDOM.VALUE(0,100));
```

11.10.3 DBMS_OUTPUT

Related Interfaces

[Table 11-21](#) provides all interfaces supported by the **DBMS_OUTPUT** package.

Table 11-21 DBMS_OUTPUT

API	Description
DBMS_OUTPUT.PUT_LINE	Outputs the specified text. The text length cannot exceed 32,767 bytes.
DBMS_OUTPUT.PUT	Outputs the specified text to the front of the specified text without adding a line break. The text length cannot exceed 32,767 bytes.
DBMS_OUTPUT.ENABLE	Sets the buffer area size. If this interface is not specified, the maximum buffer size is 20,000 bytes and the minimum buffer size is 2,000 bytes. If the specified buffer size is less than 2,000 bytes, the default minimum buffer size is applied.

- DBMS_OUTPUT.PUT_LINE

The **PUT_LINE** procedure writes a row of text carrying a line end symbol in the buffer. The **DBMS_OUTPUT.PUT_LINE** function prototype is:

```
DBMS_OUTPUT.PUT_LINE (  
item IN VARCHAR2);
```

Table 11-22 DBMS_OUTPUT.PUT_LINE interface parameters

Parameter	Description
item	Specifies the text that was written to the buffer.

- DBMS_OUTPUT.PUT

The stored procedure **PUT** outputs the specified text to the front of the specified text without adding a linefeed. The **DBMS_OUTPUT.PUT** function prototype is:

```
DBMS_OUTPUT.PUT (  
item IN VARCHAR2);
```

Table 11-23 DBMS_OUTPUT.PUT interface parameters

Parameter	Description
item	Specifies the text that was written to the specified text.

- DBMS_OUTPUT.ENABLE

The stored procedure **ENABLE** sets the output buffer size. If the size is not specified, it contains a maximum of 20,000 bytes. The **DBMS_OUTPUT.ENABLE** function prototype is:

```
DBMS_OUTPUT.ENABLE (  
buf IN INTEGER);
```

Table 11-24 DBMS_OUTPUT.ENABLE interface parameters

Parameter	Description
buf	Sets the buffer area size.

Examples

```
BEGIN
  DBMS_OUTPUT.ENABLE(50);
  DBMS_OUTPUT.PUT ('hello, ');
  DBMS_OUTPUT.PUT_LINE('database!');-- Displaying "hello, database!"
END;
/
```

11.10.4 UTL_RAW

Related Interfaces

[Table 11-25](#) provides all interfaces supported by the **UTL_RAW** package.

Table 11-25 UTL_RAW

API	Description
UTL_RAW.CAST_FROM_BINARY_INTEGER	Converts an INTEGER type value to a binary representation (RAW type).
UTL_RAW.CAST_TO_BINARY_INTEGER	Converts a binary representation (RAW type) to an INTEGER type value.
UTL_RAW.LENGTH	Obtains the length of the RAW type object.
UTL_RAW.CAST_TO_RAW	Converts a VARCHAR2 type value to a binary expression (RAW type).

NOTICE

The external representation of the RAW type data is hexadecimal and its internal storage form is binary. For example, the representation of the **RAW** type data **11001011** is 'CB'. The input of the actual type conversion is 'CB'.

- **UTL_RAW.CAST_FROM_BINARY_INTEGER**
The stored procedure **CAST_FROM_BINARY_INTEGER** converts an **INTEGER** type value to a binary representation (**RAW** type).

The **UTL_RAW.CAST_FROM_BINARY_INTEGER** function prototype is:

```
UTL_RAW.CAST_FROM_BINARY_INTEGER (
  n          IN INTEGER,
  endianness IN INTEGER)
RETURN RAW;
```

Table 11-26 UTL_RAW.CAST_FROM_BINARY_INTEGER interface parameters

Parameter	Description
n	Specifies the INTEGER type value to be converted to the RAW type.
endianess	Specifies the INTEGER type value 1 or 2 of the byte sequence. (1 indicates BIG_ENDIAN and 2 indicates LITTLE-ENDIAN .)

- UTL_RAW.CAST_TO_BINARY_INTEGER

The stored procedure CAST_TO_BINARY_INTEGER converts an INTEGER type value in a binary representation (RAW type) to the INTEGER type.

The UTL_RAW.CAST_TO_BINARY_INTEGER function prototype is:

```
UTL_RAW.CAST_TO_BINARY_INTEGER (  
r      IN RAW,  
endianess IN INTEGER)  
RETURN BINARY_INTEGER;
```

Table 11-27 UTL_RAW.CAST_TO_BINARY_INTEGER interface parameters

Parameter	Description
r	Specifies an INTEGER type value in a binary representation (RAW type).
endianess	Specifies the INTEGER type value 1 or 2 of the byte sequence. (1 indicates BIG_ENDIAN and 2 indicates LITTLE-ENDIAN .)

- UTL_RAW.LENGTH

The stored procedure LENGTH returns the length of a RAW type object.

The UTL_RAW.LENGTH function prototype is:

```
UTL_RAW.LENGTH(  
r      IN RAW)  
RETURN INTEGER;
```

Table 11-28 UTL_RAW.LENGTH interface parameters

Parameter	Description
r	Specifies a RAW type object.

- UTL_RAW.CAST_TO_RAW

The stored procedure CAST_TO_RAW converts a VARCHAR2 type object to the RAW type.

The UTL_RAW.CAST_TO_RAW function prototype is:

```
UTL_RAW.CAST_TO_RAW(  
c      IN VARCHAR2)  
RETURN RAW;
```

Table 11-29 UTL_RAW.CAST_TO_RAW interface parameters

Parameter	Description
c	Specifies a VARCHAR2 type object to be converted.

Example

Perform operations on RAW data in a stored procedure:

```
CREATE OR REPLACE PROCEDURE proc_raw
AS
str varchar2(100) := 'abcdef';
source raw(100);
amount integer;
BEGIN
source := utl_raw.cast_to_raw(str);--Convert the type.
amount := utl_raw.length(source);--Obtain the length.
dbms_output.put_line(amount);
END;
/
```

Call the stored procedure:

```
CALL proc_raw();
```

11.10.5 DBMS_JOB

Related Interfaces

Table 11-30 lists all interfaces supported by the **DBMS_JOB** package.

Table 11-30 DBMS_JOB

Interface	Description
DBMS_JOB.SUBMIT	Submits a job to the job queue. The job number is automatically generated by the system.
DBMS_JOB.SUBMIT_NODE	Submits a job to the job queue. The execution node is specified by the user, and the job number is automatically generated by the system.
DBMS_JOB.ISUBMIT	Submits a job to the job queue. The job number is specified by the user.
DBMS_JOB.REMOVE	Removes a job from the job queue by job number.
DBMS_JOB.BROKEN	Disables or enables job execution.
DBMS_JOB.CHANGE	Modifies user-definable attributes of a job, including the job description, next execution time, and execution interval.
DBMS_JOB.WHAT	Modifies the job description of a job.

Interface	Description
DBMS_JOB.NEXT_DATE	Modifies the next execution time of a job.
DBMS_JOB.INTERVAL	Modifies the execution interval of a job.
DBMS_JOB.CHANGE_OWNER	Modifies the owner of a job.
DBMS_JOB.CHANGE_NODE	Modifies the execution node of the scheduled task.

- DBMS_JOB.SUBMIT

The stored procedure **SUBMIT** submits a job provided by the system.

A prototype of the DBMS_JOB.SUBMIT function is as follows:

```
DBMS_JOB.SUBMIT(  
  what      IN TEXT,  
  next_date IN TIMESTAMP DEFAULT sysdate,  
  job_interval IN TEXT DEFAULT 'null',  
  job       OUT INTEGER);
```

 NOTE

When a job is created (using DBMS_JOB), the system binds the current database and the username to the job by default. This function can be invoked by using **call** or **select**. If you invoke this function by using **select**, there is no need to specify output parameters. To invoke this function within a stored procedure, use **perform**.

Table 11-31 DBMS_JOB.SUBMIT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
what	text	IN	No	SQL statement to be executed. One or multiple DMLs, anonymous blocks, and SQL statements that invoke stored procedures, or all three combined are supported.
next_date	timestamp	IN	No	Specifies the next time the job will be executed. The default value is the current system time (sysdate). If the specified time has past, the job is executed at the time it is submitted.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
interval	text	IN	Yes	Calculates the next time to execute the job. It can be an interval expression, or sysdate followed by a numeric value, for example, sysdate+1.0/24 . If this parameter is left blank or set to null , the job will be executed only once, and the job status will change to 'd' afterward.
job	integer	OUT	No	Specifies the job number. The value ranges from 1 to 32767. When dbms.submit is invoked using select , this parameter can be skipped.

For example:

```
select DBMS_JOB.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1');

select DBMS_JOB.SUBMIT('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1.0/24');

CALL DBMS_JOB.SUBMIT('INSERT INTO T_JOB VALUES(1); call pro_1(); call pro_2();',
add_months(to_date('201701','yyyymm'),1), 'date_trunc("day",SYSDATE) + 1 +(8*60+30.0)/
(24*60)' ,jobid);
```

- **DBMS_JOB.SUBMIT_NODE**

The stored procedure **SUBMIT** submits a job provided by the system. The execution node is specified by the user. This interface is supported only by clusters of version 8.3.0 or later.

The prototype of the DBMS_JOB.SUBMIT_NODE function is:

```
DBMS_JOB.SUBMIT_NODE(
what      IN TEXT,
next_date IN TIMESTAMP DEFAULT sysdate,
job_interval IN TEXT DEFAULT 'null',
job_node   IN TEXT DEFAULT NULL,
job       OUT INTEGER);
```

Table 11-32 DBMS_JOB.SUBMIT_NODE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
what	text	IN	No	Specifies the SQL statement to be executed. One or multiple DMLs, anonymous blocks, and SQL statements that invoke stored procedures, or all three combined are supported.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
next_date	timestamp	IN	No	Specifies the next time the job will be executed. The default value is the current system time (sysdate). If the specified time has past, the job is executed at the time it is submitted.
interval	text	IN	Yes	Calculates the next time to execute the job. It can be an interval expression, or sysdate followed by a numeric value, for example, sysdate+1.0/24 . If this parameter is left blank or set to null , the job will be executed only once, and the job status will change to 'd' afterward.
node	text	IN	Yes	Specifies the name of the job execution node.
job	integer	OUT	No	Specifies the job number. The value ranges from 1 to 32767. When dbms.submit is invoked using select , this parameter can be skipped.

For example:

```
select DBMS_JOB.SUBMIT_NODE('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1','coordinator1');
```

```
select DBMS_JOB.SUBMIT_NODE('call pro_xxx();', to_date('20180101','yyyymmdd'),'sysdate+1.0/24');
```

```
CALL DBMS_JOB.SUBMIT('INSERT INTO T_JOB VALUES(1); call pro_1(); call pro_2();',  
add_months(to_date('201701','yyyymm'),1), 'date_trunc("day",SYSDATE) + 1 +(8*60+30.0)/(24*60)',  
'coordinator1', :jobid);
```

- **DBMS_JOB.ISUBMIT**

ISUBMIT has the same syntax function as **SUBMIT**, but the first parameter of **ISUBMIT** is an input parameter, that is, a specified job number. In contrast, that last parameter of **SUBMIT** is an output parameter, indicating the job number automatically generated by the system.

For example:

```
CALL dbms_job.isubmit(101, 'insert_msg_statistic1;', sysdate, 'sysdate+3.0/24');
```

NOTICE

The pgstats persistence function of DWS writes the statistics in the memory to the **pg_stat_object** system catalog. If the cluster version is 9.1.0.100 or later, **1** is used as **job_id**. If an earlier cluster version is upgraded to 9.1.0.100 or later and **pg_job** contains tasks, an unoccupied **job_id** is used as the ID of the persistence task. Therefore, when using the **dbms_job.isubmit** interface, ensure that the ID is different from the ID of an existing pgstats persistence task. Otherwise, the task registration fails.

- **DBMS_JOB.REMOVE**

The stored procedure **REMOVE** deletes a specified job.

A prototype of the DBMS_JOB.REMOVE function is as follows:

```
REMOVE(job IN INTEGER);
```

Table 11-33 DBMS_JOB.REMOVE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

For example:

```
CALL dbms_job.remove(101);
```

- **DBMS_JOB.BROKEN**

The stored procedure **BROKEN** sets the broken flag of a job.

A prototype of the DBMS_JOB.BROKEN function is as follows:

```
DBMS_JOB.BROKEN(  
  job      IN  INTEGER,  
  broken   IN  BOOLEAN,  
  next_date IN  TIMESTAMP DEFAULT sysdate);
```

Table 11-34 DBMS_JOB.BROKEN interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
broken	boolean	IN	No	Specifies the status flag, true for broken and false for not broken. Setting this parameter to true or false updates the current job. If the parameter is left blank, the job status remains unchanged.
next_date	timestamp	IN	Yes	Specifies the next execution time. The default is the current system time. If broken is set to true , next_date is updated to '4000-1-1'. If broken is false and next_date is not empty, next_date is updated for the job. If next_date is empty, it will not be updated. This parameter can be omitted, and its default value will be used in this case.

For example:

```
CALL dbms_job.broken(101,true);  
CALL dbms_job.broken(101,false,sysdate);
```

- **DBMS_JOB.CHANGE**

The stored procedure **CHANGE** modifies user-definable attributes of a job, including the job content, next-execution time, and execution interval.

A prototype of the DBMS_JOB.CHANGE function is as follows:

```
DBMS_JOB.CHANGE(  
  job      IN  INTEGER,  
  what     IN  TEXT,  
  next_date IN  TIMESTAMP,  
  interval IN  TEXT);
```

Table 11-35 DBMS_JOB.CHANGE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
what	text	IN	Yes	Specifies the name of the stored procedure or SQL statement block that is executed. If this parameter is left blank, the system does not update the what parameter for the specified job. Otherwise, the system updates the what parameter for the specified job.
next_date	timestamp	IN	Yes	Specifies the next execution time. If this parameter is left blank, the system does not update the next_date parameter for the specified job. Otherwise, the system updates the next_date parameter for the specified job.
interval	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left blank, the system does not update the interval parameter for the specified job. Otherwise, the system updates the interval parameter for the specified job after necessary validity check. If this parameter is set to null , the job will be executed only once, and the job status will change to 'd' afterward.

For example:

```
CALL dbms_job.change(101, 'call userproc();', sysdate, 'sysdate + 1.0/1440');  
CALL dbms_job.change(101, 'insert into tbl_a values(sysdate);', sysdate, 'sysdate + 1.0/1440');
```

- **DBMS_JOB.WHAT**

The stored procedure **WHAT** modifies the procedures to be executed by a specified job.

A prototype of the DBMS_JOB.WHAT function is as follows:

```
DBMS_JOB.WHAT(  
  job      IN   INTEGER,  
  what     IN   TEXT);
```

Table 11-36 DBMS_JOB.WHAT interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
what	text	IN	No	Specifies the name of the stored procedure or SQL statement block that is executed.

 NOTE

- If the value specified by the **what** parameter is one or multiple executable SQL statements, program blocks, or stored procedures, this procedure can be executed successfully; otherwise, it will fail to be executed.
- If the **what** parameter is a simple statement such as insert and update, a schema name must be added in front of the table name.

For example:

```
CALL dbms_job.what(101, 'call userproc();');  
CALL dbms_job.what(101, 'insert into tbl_a values(sysdate);');
```

- **DBMS_JOB.NEXT_DATE**

The stored procedure **NEXT_DATE** modifies the next-execution time attribute of a job.

A prototype of the DBMS_JOB.NEXT_DATE function is as follows:

```
DBMS_JOB.NEXT_DATE(  
  job      IN  INTEGER,  
  next_date IN  TIMESTAMP);
```

Table 11-37 DBMS_JOB.NEXT_DATE interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
next_date	timestamp	IN	No	Specifies the next execution time.

 NOTE

If the specified **next_date** value is earlier than the current date, the job is executed once immediately.

For example:

```
CALL dbms_job.next_date(101,sysdate);
```

- DBMS_JOB.INTERVAL

The stored procedure **INTERVAL** modifies the execution interval attribute of a job.

A prototype of the DBMS_JOB.INTERVAL function is as follows:

```
DBMS_JOB.INTERVAL(  
job          IN  INTEGER,  
interval     IN  TEXT);
```

Table 11-38 DBMS_JOB.INTERVAL interface parameters

Parameter	Type	Input / Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
interval	text	IN	Yes	Specifies the time expression for calculating the next time the job will be executed. If this parameter is left blank or set to null , the job will be executed only once, and the job status will change to 'd' afterward. interval must be a valid time or interval type.

For example:

```
CALL dbms_job.interval(101, 'sysdate + 1.0/1440');
```

 NOTE

For a job that is currently running (that is, **job_status** is 'r'), it is not allowed to use **remove**, **change**, **next_date**, **what**, or **interval** to delete or modify job parameters.

- DBMS_JOB.CHANGE_OWNER

The stored procedure **CHANGE_OWNER** modifies the owner of a job.

A prototype of the DBMS_JOB.CHANGE_OWNER function is as follows:

```
DBMS_JOB.CHANGE_OWNER(  
job          IN  INTEGER,  
new_owner    IN  NAME);
```

Table 11-39 DBMS_JOB.CHANGE_OWNER interface parameters

Parameter	Type	Input/ Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
new_owner	name	IN	No	Specifies the new username.

For example:

```
CALL dbms_job.change_owner(101, 'alice');
```

- **DBMS_JOB.CHANGE_NODE**

The stored procedure **CHANGE_NODE** modifies the execution node of the scheduled task. This interface is supported only by clusters of version 8.3.0 or later.

A prototype of the DBMS_JOB.CHANGE_NODE function is:

```
DBMS_JOB.CHANGE_NODE(  
  job      IN  INTEGER,  
  new_node IN  text);
```

Table 11-40 DBMS_JOB.CHANGE_OWNER interface parameters

Parameter	Type	Input/Output Parameter	Can Be Empty	Description
job	integer	IN	No	Specifies the job number.
new_node	text	IN	No	Specifies the new execution node.

For example:

```
CALL dbms_job.change_node(101, 'coordinator2');
```

Constraints

1. After a new job is created, this job belongs to the current coordinator only, that is, this job can be scheduled and executed only on the current coordinator. Other coordinators will not schedule or execute this job. All coordinators can query, modify, and delete jobs created on other CNs.
2. Create, update, and delete jobs only using the procedures provided by the DBMS_JOB package. These procedures synchronize job information between different CNs and associate primary keys between the **pg_jobs** tables. If you use DML statements to add, delete, or modify records in the **pg_jobs** table, job information will become inconsistent between CNs and system tables may fail to be associated, compromising internal job management.
3. Each user-created task is bound to a CN. If the automatic migration function is not enabled, task statuses cannot be updated in real time when the CN is

faulty during task execution. When a CN fails, all jobs on this CN cannot be scheduled or executed until the CN is restored manually. Enable the automatic migration function on CNs, so that jobs on the faulty CN will be migrated to other CNs for scheduling.

4. For each job, the hosting CN updates the real-time job information (including the job status, last execution start time, last execution end time, next execution start time, the number of execution failures if any) to the **pg_jobs** table, and synchronizes the information to other CNs, ensuring consistent job information between different CNs. In the case of CN failures, job information synchronization is reattempted by the hosting CNs, which increases job execution time. Although job information fails to be synchronized between CNs, job information can still be properly updated in the **pg_jobs** table on the hosting CNs, and jobs can be executed successfully. After a CN recovers, job information such as job execution time and status in its **pg_jobs** table may be incorrect and will be updated only after the jobs are executed again on related CNs.
5. For each job, a thread is established to execute it. If multiple jobs are triggered concurrently as scheduled, the system will need some time to start the required threads, resulting in a latency of 0.1 ms in job execution.
6. The length of the SQL statement to be executed in a job is limited. The maximum length is 8 KB.

11.10.6 DBMS_SQL

Related Interfaces

Table 11-41 lists interfaces supported by the **DBMS_SQL** package.

Table 11-41 DBMS_SQL

API	Description
DBMS_SQL.OPEN_CURSOR	Opens a cursor.
DBMS_SQL.CLOSE_CURSOR	Closes an open cursor.
DBMS_SQL.PARSE	Transmits a group of SQL statements to a cursor. Currently, only the SELECT statement is supported.
DBMS_SQL.EXECUTE	Performs a set of dynamically defined operations on the cursor.
DBMS_SQL.FETCH_ROWS	Reads a row of cursor data.
DBMS_SQL.DEFINE_COLUMN	Dynamically defines a column.
DBMS_SQL.DEFINE_COLUMN_CHAR	Dynamically defines a column of the CHAR type.
DBMS_SQL.DEFINE_COLUMN_INT	Dynamically defines a column of the INT type.

API	Description
DBMS_SQL.DEFINE_COLUMN_LONG	Dynamically defines a column of the LONG type.
DBMS_SQL.DEFINE_COLUMN_RAW	Dynamically defines a column of the RAW type.
DBMS_SQL.DEFINE_COLUMN_TEXT	Dynamically defines a column of the TEXT type.
DBMS_SQL.DEFINE_COLUMN_UNKNOWN	Dynamically defines a column of an unknown type.
DBMS_SQL.COLUMN_VALUE	Reads a dynamically defined column value.
DBMS_SQL.COLUMN_VALUE_CHAR	Reads a dynamically defined column value of the CHAR type.
DBMS_SQL.COLUMN_VALUE_INT	Reads a dynamically defined column value of the INT type.
DBMS_SQL.COLUMN_VALUE_LONG	Reads a dynamically defined column value of the LONG type.
DBMS_SQL.COLUMN_VALUE_RAW	Reads a dynamically defined column value of the RAW type.
DBMS_SQL.COLUMN_VALUE_TEXT	Reads a dynamically defined column value of the TEXT type.
DBMS_SQL.COLUMN_VALUE_UNKNOWN	Reads a dynamically defined column value of an unknown type.
DBMS_SQL.IS_OPEN	Checks whether a cursor is opened.

 NOTE

- You are advised to use **dbms_sql.define_column** and **dbms_sql.column_value** to define columns.
- If the size of the result set is greater than the value of **work_mem**, the result set will be flushed to disk. The value of **work_mem** must be no greater than 512 MB.
- **DBMS_SQL.OPEN_CURSOR**
This function opens a cursor and is the prerequisite for the subsequent **dbms_sql** operations. This function does not transfer any parameter. It automatically generates cursor IDs in an ascending order and returns values to integer variables.

The function prototype of **DBMS_SQL.OPEN_CURSOR** is:

```
DBMS_SQL.OPEN_CURSOR (  
)  
RETURN INTEGER;
```

- **DBMS_SQL.CLOSE_CURSOR**

This function closes a cursor. It is the end of each `dbms_sql` operation. If this function is not invoked when the stored procedure ends, the memory is still occupied by the cursor. Therefore, remember to close a cursor when you do not need to use it. If an exception occurs, the stored procedure exits but the cursor is not closed. Therefore, you are advised to include this interface in the exception handling of the stored procedure.

The function prototype of **DBMS_SQL.CLOSE_CURSOR** is:

```
DBMS_SQL.CLOSE_CURSOR (  
  cursorid    IN INTEGER  
)  
RETURN INTEGER;
```

Table 11-42 DBMS_SQL.CLOSE_CURSOR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be closed

- **DBMS_SQL.PARSE**

This function parses the query statement of a given cursor. The input query statement is executed immediately. Currently, only the **SELECT** query statement can be parsed. The statement parameters can be transferred only through the TEXT type. The length cannot exceed 1 GB.

The function prototype of **DBMS_SQL.PARSE** is:

```
DBMS_SQL.PARSE (  
  cursorid    IN INTEGER,  
  query_string IN TEXT,  
  label       IN INTEGER  
)  
RETURN BOOLEAN;
```

Table 11-43 DBMS_SQL.PARSE interface parameters

Parameter Name	Description
cursorid	ID of the cursor whose query statement is parsed
query_string	Query statements to be parsed
language_flag	Version language number. Currently, only 1 is supported.

- **DBMS_SQL.EXECUTE**

This function executes a given cursor. This function receives a cursor ID. The obtained data after is used for subsequent operations. Currently, only the **SELECT** query statement can be executed.

The function prototype of **DBMS_SQL.EXECUTE** is:

```
DBMS_SQL.EXECUTE(  
  cursorid    IN INTEGER,  
)  
RETURN INTEGER;
```

Table 11-44 DBMS_SQL.EXECUTE interface parameters

Parameter Name	Description
cursorid	ID of the cursor whose query statement is parsed

- DBMS_SQL.FETCH_ROWS

This function returns the number of data rows that meet query conditions. Each time the interface is executed, the system obtains a set of new rows until all data is read.

The prototype of the **DBMS_SQL.FETCH_ROWS** function is as follows:

```
DBMS_SQL.FETCH_ROWS(  
  cursorid IN INTEGER,  
)  
RETURN INTEGER;
```

Table 11-45 DBMS_SQL.FETCH_ROWS interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed

- DBMS_SQL.DEFINE_COLUMN

This function defines columns returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN** is:

```
DBMS_SQL.DEFINE_COLUMN(  
  cursorid IN INTEGER,  
  position IN INTEGER,  
  column_ref IN ANYELEMENT,  
  column_size IN INTEGER default 1024  
)  
RETURN INTEGER;
```

Table 11-46 DBMS_SQL.DEFINE_COLUMN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_ref	Variable of any type. You can select an appropriate interface to dynamically define columns based on variable types.
column_size	Length of a defined column

- DBMS_SQL.DEFINE_COLUMN_CHAR

This function defines columns of the CHAR type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_CHAR** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(  
  cursorid   IN INTEGER,  
  position   IN INTEGER,  
  column     IN TEXT,  
  column_size IN INTEGER  
)  
RETURN INTEGER;
```

Table 11-47 DBMS_SQL.DEFINE_COLUMN_CHAR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Parameter to be defined
column_size	Length of a dynamically defined column

- **DBMS_SQL.DEFINE_COLUMN_INT**

This function defines columns of the INT type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_INT** is:

```
DBMS_SQL.DEFINE_COLUMN_INT(  
  cursorid   IN INTEGER,  
  position   IN INTEGER  
)  
RETURN INTEGER;
```

Table 11-48 DBMS_SQL.DEFINE_COLUMN_INT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- **DBMS_SQL.DEFINE_COLUMN_LONG**

This function defines columns of a long type (not LONG) returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type. The maximum size of a long column is 1 GB.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_LONG** is:

```
DBMS_SQL.DEFINE_COLUMN_LONG(  
  cursorid IN INTEGER,  
  position IN INTEGER  
)  
RETURN INTEGER;
```

Table 11-49 DBMS_SQL.DEFINE_COLUMN_LONG interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- DBMS_SQL.DEFINE_COLUMN_RAW

This function defines columns of the RAW type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_RAW** is:

```
DBMS_SQL.DEFINE_COLUMN_RAW(  
  cursorid IN INTEGER,  
  position IN INTEGER,  
  column IN BYTEA,  
  column_size IN INTEGER  
)  
RETURN INTEGER;
```

Table 11-50 DBMS_SQL.DEFINE_COLUMN_RAW interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Parameter of the RAW type
column_size	Column length

- DBMS_SQL.DEFINE_COLUMN_TEXT

This function defines columns of the TEXT type returned from a given cursor and can be used only for the cursors defined by **SELECT**. The defined columns are identified by the relative positions in the query list. The data type of the input variable determines the column type.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_TEXT** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(  
  cursorid IN INTEGER,  
  position IN INTEGER,  
  max_size IN INTEGER  
)  
RETURN INTEGER;
```

Table 11-51 DBMS_SQL.DEFINE_COLUMN_TEXT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
max_size	Maximum length of the defined TEXT type

- DBMS_SQL.DEFINE_COLUMN_UNKNOWN

This function processes columns of unknown data types returned from a given cursor and is used only for the system to report an error and exist when the type cannot be identified.

The function prototype of **DBMS_SQL.DEFINE_COLUMN_UNKNOWN** is:

```
DBMS_SQL.DEFINE_COLUMN_CHAR(  
  cursorid    IN INTEGER,  
  position    IN INTEGER,  
  column      IN TEXT  
)  
RETURN INTEGER;
```

Table 11-52 DBMS_SQL.DEFINE_COLUMN_UNKNOWN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column	Dynamically defined parameter

- DBMS_SQL.COLUMN_VALUE

This function returns the cursor element value specified by a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE** is:

```
DBMS_SQL.COLUMN_VALUE(  
  cursorid    IN INTEGER,  
  position    IN INTEGER,  
  column_value INOUT ANYELEMENT  
)  
RETURN ANYELEMENT;
```

Table 11-53 DBMS_SQL.COLUMN_VALUE interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

Parameter Name	Description
column_value	Return value of a defined column

- DBMS_SQL.COLUMN_VALUE_CHAR

This function returns the value of the CHAR type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_CHAR** is:

```
DBMS_SQL.COLUMN_VALUE_CHAR(  
  cursorid      IN   INTEGER,  
  position      IN   INTEGER,  
  column_value  INOUT CHARACTER,  
  err_num       INOUT NUMERIC default 0,  
  actual_length INOUT INTEGER default 1024  
)  
RETURN RECORD;
```

Table 11-54 DBMS_SQL.COLUMN_VALUE_CHAR interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_value	Return value
err_num	Error No. It is an output parameter and the argument must be a variable. Currently, the output value is -1 regardless of the argument.
actual_length	Length of a return value

- DBMS_SQL.COLUMN_VALUE_INT

This function returns the value of the INT type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS. The function prototype of **DBMS_SQL.COLUMN_VALUE_INT** is:

```
DBMS_SQL.COLUMN_VALUE_INT(  
  cursorid      IN   INTEGER,  
  position      IN   INTEGER  
)  
RETURN INTEGER;
```

Table 11-55 DBMS_SQL.COLUMN_VALUE_INT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- DBMS_SQL.COLUMN_VALUE_LONG

This function returns the value of a long type (not LONG or BIGINT) in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_LONG** is:

```
DBMS_SQL.COLUMN_VALUE_LONG(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER,  
  length        IN  INTEGER,  
  off_set       IN  INTEGER,  
  column_value  INOUT TEXT,  
  actual_length INOUT INTEGER default 1024  
)  
RETURN RECORD;
```

Table 11-56 DBMS_SQL.COLUMN_VALUE_LONG interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
length	Length of a return value
off_set	Start position of a return value
column_value	Return value
actual_length	Length of a return value

- DBMS_SQL.COLUMN_VALUE_RAW

This function returns the value of the RAW type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_RAW** is:

```
DBMS_SQL.COLUMN_VALUE_RAW(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER,  
  column_value  INOUT BYTEA,  
  err_num       INOUT NUMERIC default 0,  
  actual_length INOUT INTEGER default 1024  
)  
RETURN RECORD;
```

Table 11-57 DBMS_SQL.COLUMN_VALUE_RAW interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_value	Returned column value

Parameter Name	Description
err_num	Error No. It is an output parameter and the argument must be a variable. Currently, the output value is -1 regardless of the argument.
actual_length	Length of a return value. The value longer than this length will be truncated.

- **DBMS_SQL.COLUMN_VALUE_TEXT**

This function returns the value of the TEXT type in a specified position of a cursor and accesses the data obtained by DBMS_SQL.FETCH_ROWS.

The function prototype of **DBMS_SQL.COLUMN_VALUE_TEXT** is:

```
DBMS_SQL.COLUMN_VALUE_TEXT(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER  
)  
RETURN TEXT;
```

Table 11-58 DBMS_SQL.COLUMN_VALUE_TEXT interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query

- **DBMS_SQL.COLUMN_VALUE_UNKNOWN**

This function returns the value of an unknown type in a specified position of a cursor. This is an error handling interface when the type is not unknown.

The function prototype of **DBMS_SQL.COLUMN_VALUE_UNKNOWN** is:

```
DBMS_SQL.COLUMN_VALUE_UNKNOWN(  
  cursorid      IN  INTEGER,  
  position      IN  INTEGER,  
  COLUMN_TYPE    IN  TEXT  
)  
RETURN TEXT;
```

Table 11-59 DBMS_SQL.COLUMN_VALUE_UNKNOWN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be executed
position	Position of a dynamically defined column in the query
column_type	Returned parameter type

- **DBMS_SQL.IS_OPEN**

This function returns the status of a cursor: **open**, **parse**, **execute**, or **define**. The value is **TRUE**. If the status is unknown, an error is reported. In other cases, the value is **FALSE**.

The function prototype of **DBMS_SQL.IS_OPEN** is:

```
DBMS_SQL.IS_OPEN(  
    cursorid          IN   INTEGER  
)  
RETURN BOOLEAN;
```

Table 11-60 DBMS_SQL.IS_OPEN interface parameters

Parameter Name	Description
cursorid	ID of the cursor to be queried

Examples

```
--Perform operations on raw data in a stored procedure.  
create or replace procedure pro_dbms_sql_all_02(in_raw raw,v_in int,v_offset int)  
as  
    cursorid int;  
    v_id int;  
    v_info bytea :=1;  
    query varchar(2000);  
    execute_ret int;  
    define_column_ret_raw bytea :=1;  
    define_column_ret int;  
begin  
    drop table if exists pro_dbms_sql_all_tb1_02 ;  
    create table pro_dbms_sql_all_tb1_02(a int ,b blob);  
    insert into pro_dbms_sql_all_tb1_02 values(1,HEXTORAW('DEADBEEF'));  
    insert into pro_dbms_sql_all_tb1_02 values(2,in_raw);  
    query := 'select * from pro_dbms_sql_all_tb1_02 order by 1';  
    --Open a cursor.  
    cursorid := dbms_sql.open_cursor();  
    --Compile the cursor.  
    dbms_sql.parse(cursorid, query, 1);  
    --Define a column.  
    define_column_ret:= dbms_sql.define_column(cursorid,1,v_id);  
    define_column_ret_raw:= dbms_sql.define_column_raw(cursorid,2,v_info,10);  
    --Execute the cursor.  
    execute_ret := dbms_sql.execute(cursorid);  
    loop  
    exit when (dbms_sql.fetch_rows(cursorid) <= 0);  
    --Obtain values.  
    dbms_sql.column_value(cursorid,1,v_id);  
    dbms_sql.column_value_raw(cursorid,2,v_info,v_in,v_offset);  
    --Output the result.  
    dbms_output.put_line('id: || v_id || ' info:' || v_info);  
    end loop;  
    --Close the cursor.  
    dbms_sql.close_cursor(cursorid);  
end;  
/  
--Invoke the stored procedure.  
call pro_dbms_sql_all_02(HEXTORAW('DEADBEEF'),0,1);  
  
--Delete the stored procedure.  
DROP PROCEDURE pro_dbms_sql_all_02;
```

11.11 DWS Stored Procedure Debugging

Syntax

RAISE has the following five syntax formats:

Figure 11-34 raise_format::=

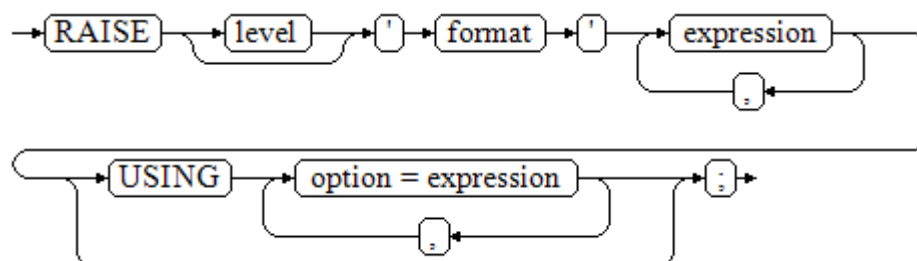


Figure 11-35 raise_condition::=

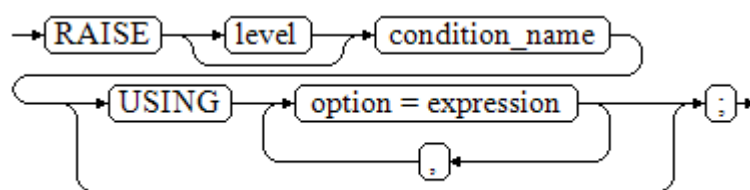


Figure 11-36 raise_sqlstate::=

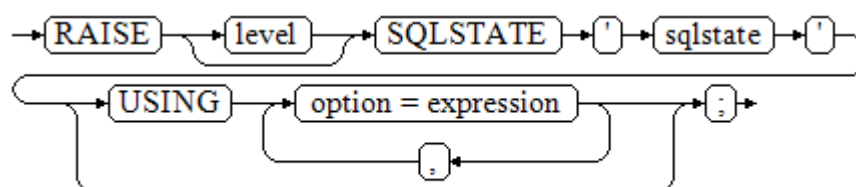


Figure 11-37 raise_option::=

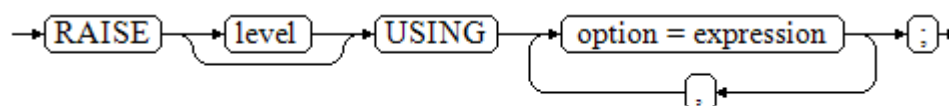
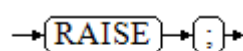


Figure 11-38 raise::=



Parameter description:

- The level option is used to specify the error level, that is, **DEBUG**, **LOG**, **INFO**, **NOTICE**, **WARNING**, or **EXCEPTION** (default). **EXCEPTION** reports an error that normally terminates the current transaction and the others only generate information at their levels. The [log_min_messages](#) and [client_min_messages](#) parameters control whether the error messages of specific levels are reported to the client and are written to the server log.
- **format**: specifies the error message text to be reported, a format character string. The format character string can be appended with an expression for insertion to the message text. In a format character string, **%** is replaced by the parameter value attached to format and **%%** is used to print **%**. For example:

```
--v_job_id replaces % in the character string.  
RAISE NOTICE 'Calling cs_create_job(%)',v_job_id;
```
- **option = expression**: inserts additional information to an error report. The keyword option can be **MESSAGE**, **DETAIL**, **HINT**, or **ERRCODE**, and each expression can be any character string.
 - **MESSAGE**: specifies the error message text. This option cannot be used in a **RAISE** statement that contains a format character string in front of **USING**.
 - **DETAIL**: specifies detailed information of an error.
 - **HINT**: prints hint information.
 - **ERRCODE**: designates an error code (SQLSTATE) to a report. A condition name or a five-character SQLSTATE error code can be used.
- **condition_name**: specifies the condition name corresponding to the error code.
- **sqlstate**: specifies the error code.

If neither a condition name nor an **SQLSTATE** is designated in a **RAISE EXCEPTION** command, the **RAISE EXCEPTION (P0001)** is used by default. If no message text is designated, the condition name or SQLSTATE is used as the message text by default.

NOTICE

If the **SQLSTATE** designates an error code, the error code is not limited to a defined error code. It can be any error code containing five digits or ASCII uppercase rather than **00000**. Avoid using error codes that end with three zeros as they are type codes and can be captured by the entire category.

NOTE

The syntax described in [Figure 11-38](#) does not append any parameter. This form is used only for the **EXCEPTION** statement in a **BEGIN** block so that the error can be re-processed.

Examples

Display error and hint information when a transaction terminates:

```
CREATE OR REPLACE PROCEDURE proc_raise1(user_id in integer)  
AS  
BEGIN  
RAISE EXCEPTION 'Noexistence ID --> %',user_id USING HINT = 'Please check your user ID';
```

```
END;  
/  
  
call proc_raise1(300011);  
  
-- Execution result:  
ERROR: Noexistence ID --> 300011  
HINT: Please check your user ID
```

Two methods are available for setting **SQLSTATE**:

```
CREATE OR REPLACE PROCEDURE proc_raise2(user_id in integer)  
AS  
BEGIN  
  RAISE 'Duplicate user ID: %',user_id USING ERRCODE = 'unique_violation';  
END;  
/  
  
\set VERBOSITY verbose  
call proc_raise2(300011);  
  
-- Execution result:  
ERROR: Duplicate user ID: 300011  
SQLSTATE: 23505  
LOCATION: exec_stmt_raise, pl_exec.cpp:3482
```

If the main parameter is a condition name or **SQLSTATE**, the following applies:

```
RAISE division_by_zero;
```

```
RAISE SQLSTATE '22012';
```

For example:

```
CREATE OR REPLACE PROCEDURE division(div in integer, dividend in integer)  
AS  
DECLARE  
  res int;  
BEGIN  
  IF dividend=0 THEN  
    RAISE division_by_zero;  
    RETURN;  
  ELSE  
    res := div/dividend;  
    RAISE INFO 'division result: %', res;  
    RETURN;  
  END IF;  
END;  
/  
call division(3,0);  
  
-- Execution result:  
ERROR: division_by_zero
```

Alternatively:

```
RAISE unique_violation USING MESSAGE = 'Duplicate user ID: ' || user_id;
```

12 Using PostGIS Extension

12.1 PostGIS

NOTE

- The third-party software that the PostGIS Extension depends on needs to be installed separately. If you need to use PostGIS, submit a service ticket or contact technical support to submit an application.
- If the error message "ERROR: EXTENSION is not yet supported." is displayed, the PostGIS software package is not installed. Contact technical support.

DWS provides PostGIS Extension (PostGIS-2.4.2 and PostGIS-3.2.2). PostGIS Extension is a spatial database extender for PostgreSQL. It provides the following spatial information services: spatial objects, spatial indexes, spatial functions, and spatial operators. PostGIS Extension complies with the OpenGIS specifications.

In DWS, PostGIS Extension depends on the listed third-party open-source software.

- PostGIS 2.4.2 depends on the following third-party open-source software:
 - Geos 3.6.2
 - Proj 4.9.2
 - Json 0.12.1
 - Libxml2 2.7.1
 - Gdal 1.11.0
- PostGIS 3.2.2 depends on the following third-party open-source software:
 - Geos-3.11.0
 - Proj-6.0.0
 - Json 0.12.1
 - Libxml2 2.7.1
 - Sqlite3
 - protobuf-c 1.4.1
 - protobuf 3.6.1

12.2 Using PostGIS

NOTE

The third-party software that the PostGIS Extension depends on needs to be installed separately. If you need to use PostGIS, submit a service ticket or contact technical support to submit an application.

If the error message "ERROR: EXTENSION is not yet supported." is displayed, the PostGIS software package is not installed. Contact technical support.

Creating PostGIS Extension

Run the **CREATE EXTENSION** command to create PostGIS Extension.

```
CREATE EXTENSION postgis;
```

Using PostGIS Extension

Use the following function to invoke a PostGIS Extension:

```
SELECT GisFunction (Param1, Param2,.....);
```

GisFunction is the function, and **Param1** and **Param2** are function parameters. The following SQL statements are a simple illustration for PostGIS use. For details about related functions, see [PostGIS 2.4.2 Manual](#).

Example 1: Create a geometry table.

```
CREATE TABLE cities ( id integer, city_name varchar(50) );  
SELECT AddGeometryColumn('cities', 'position', 4326, 'POINT', 2);
```

Example 2: Insert geometry data.

```
INSERT INTO cities (id, position, city_name) VALUES (1,ST_GeomFromText('POINT(-9.5 23)',4326),'CityA');  
INSERT INTO cities (id, position, city_name) VALUES (2,ST_GeomFromText('POINT(-10.6 40.3)',4326),'CityB');  
INSERT INTO cities (id, position, city_name) VALUES (3,ST_GeomFromText('POINT(20.8 30.3)',4326), 'CityC');
```

Example 3: Calculate the distance between any two cities among three cities.

```
SELECT p1.city_name,p2.city_name,ST_Distance(p1.position,p2.position) FROM cities AS p1, cities AS p2  
WHERE p1.id > p2.id;
```

Deleting PostGIS Extension

Run the following command to delete PostGIS Extension from DWS:

```
DROP EXTENSION postgis [CASCADE];
```

NOTE

If PostGIS Extension is the dependee of other objects (for example, geometry tables), you need to add the **CASCADE** keyword to delete all these objects.

12.3 PostGIS Support and Constraints

Supported Data Types

In DWS, PostGIS Extension support the following data types:

- box2d
- box3d
- geometry_dump
- geometry
- geography
- raster

 NOTE

If PostGIS is used by a user other than the creator of the PostGIS, set the following GUC parameters:

```
SET behavior_compat_options = 'bind_procedure_searchpath';
```

Supported Operators and Functions

 NOTE

The **ST_Intersects** function in PostGIS uses a caching strategy that enables a high cache hit ratio for the spatial data structures of foreign tables. When there is a significant disparity in the width between the inner and foreign tables, caching the wide table's data avoid the repeated loading of large objects, leading to significant performance enhancements. Practically, leveraging [Join Order Hints](#) to designate a wide table as the foreign table ensures that the execution plan is optimized for such scenarios.

Table 12-1 Operators and functions supported by PostGIS2.4.2

Category	Function
Management functions	AddGeometryColumn, DropGeometryColumn, DropGeometryTable, PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Liblwgeom_Version, PostGIS_Lib_Build_Date, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_Scripts_Build_Date, PostGIS_Scripts_Installed, PostGIS_Version, PostGIS_LibXML_Version, PostGIS_Scripts_Released, Populate_Geometry_Columns, UpdateGeometrySRID
Geometry constructors	ST_BdPolyFromText, ST_BdMPolyFromText, ST_Box2dFromGeoHash, ST_GeogFromText, ST_GeographyFromText, ST_GeogFromWKB, ST_GeomCollFromText, ST_GeomFromEWKB, ST_GeomFromEWKT, ST_GeometryFromText, ST_GeomFromGeoHash, ST_GeomFromGML, ST_GeomFromGeoJSON, ST_GeomFromKML, ST_GMLToSQL, ST_GeomFromText, ST_GeomFromWKB, ST_LineFromMultiPoint, ST_LineFromText, ST_LineFromWKB, ST_LinestringFromWKB, ST_MakeBox2D, ST_3DMakeBox, ST_MakeEnvelope, ST_MakePolygon, ST_MakePoint, ST_MakePointM, ST_MLineFromText, ST_MPointFromText, ST_MPolyFromText, ST_Point, ST_PointFromGeoHash, ST_PointFromText, ST_PointFromWKB, ST_Polygon, ST_PolygonFromText, ST_WKBToSQL, ST_WKTToSQL

Category	Function
Geometry accessors	GeometryType, ST_Boundary, ST_CoordDim, ST_Dimension, ST_EndPoint, ST_Envelope, ST_ExteriorRing, ST_GeometryN, ST_GeometryType, ST_InteriorRingN, ST_IsClosed, ST_IsCollection, ST_IsEmpty, ST_IsRing, ST_IsSimple, ST_IsValid, ST_IsValidReason, ST_IsValidDetail, ST_M, ST_NDims, ST_NPoints, ST_NRings, ST_NumGeometries, ST_NumInteriorRings, ST_NumInteriorRing, ST_NumPatches, ST_NumPoints, ST_PatchN, ST_PointN, ST_SRID, ST_StartPoint, ST_Summary, ST_X, ST_XMax, ST_XMin, ST_Y, ST_YMax, ST_YMin, ST_Z, ST_ZMax, ST_Zmflag, ST_ZMin
Geometry editors	ST_AddPoint, ST_Affine, ST_Force2D, ST_Force3D, ST_Force3DZ, ST_Force3DM, ST_Force4D, ST_ForceCollection, ST_ForceSFS, ST_ForceRHR, ST_LineMerge, ST_CollectionExtract, ST_CollectionHomogenize, ST_Multi, ST_RemovePoint, ST_Reverse, ST_Rotate, ST_RotateX, ST_RotateY, ST_RotateZ, ST_Scale, ST_Segmentize, ST_SetPoint, ST_SetSRID, ST_SnapToGrid, ST_Snap, ST_Transform, ST_Translate, ST_TransScale
Geometry outputs	ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_AsGeoJSON, ST_AsGML, ST_AsHEXEWKB, ST_AsKML, ST_AsLatLonText, ST_AsSVG, ST_AsText, ST_AsX3D, ST_GeoHash
Operators	&&, &&&, &<, &< , &>, <<, << , =, >>, @, &>, >>, ~, ~=, <->, <#>
Spatial relationships and measurements	ST_3DClosestPoint, ST_3DDistance, ST_3DDWithin, ST_3DDFullyWithin, ST_3DIntersects, ST_3DLongestLine, ST_3DMaxDistance, ST_3DShortestLine, ST_Area, ST_Azimuth, ST_Centroid, ST_ClosestPoint, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_LineCrossingDirection, ST_Disjoint, ST_Distance, ST_HausdorffDistance, ST_MaxDistance, ST_DistanceSphere, ST_DistanceSpheroid, ST_DFullyWithin, ST_DWithin, ST_Equals, ST_HasArc, ST_Intersects, ST_Length, ST_Length2D, ST_3DLength, ST_Length_Spheroid, ST_Length2D_Spheroid, ST_3DLength_Spheroid, ST_LongestLine, ST_OrderingEquals, ST_Overlaps, ST_Perimeter, ST_Perimeter2D, ST_3DPerimeter, ST_PointOnSurface, ST_Project, ST_Relate, ST_RelateMatch, ST_ShortestLine, ST_Touches, ST_Within
Geometry processing	ST_Buffer, ST_BuildArea, ST_Collect, ST_ConcaveHull, ST_ConvexHull, ST_CurveToLine, ST_DelaunayTriangles, ST_Difference, ST_Dump, ST_DumpPoints, ST_DumpRings, ST_FlipCoordinates, ST_Intersection, ST_LineToCurve, ST_MakeValid, ST_MemUnion, ST_MinimumBoundingCircle, ST_Polygonize, ST_Node, ST_OffsetCurve, ST_RemoveRepeatedPoints, ST_SharedPaths, ST_Shift_Longitude, ST_Simplify, ST_SimplifyPreserveTopology, ST_Split, ST_SymDifference, ST_Union, ST_UnaryUnion

Category	Function
Linear referencing	ST_LineInterpolatePoint, ST_LineLocatePoint, ST_LineSubstring, ST_LocateAlong, ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint, ST_AddMeasure
Miscellaneous functions	ST_Accum, Box2D, Box3D, ST_Expand, ST_Extent, ST_3Dextent, Find_SRID, ST_MemSize
Exceptional functions	PostGIS_AddBBox, PostGIS_DropBBox, PostGIS_HasBBox
Raster Management Functions	AddRasterConstraints, DropRasterConstraints, AddOverviewConstraints, DropOverviewConstraints, PostGIS_GDAL_Version, PostGIS_Raster_Lib_Build_Date, PostGIS_Raster_Lib_Version, and ST_GDALDrivers, and UpdateRasterSRID
Raster Constructors	ST_AddBand, ST_AsRaster, ST_Band, ST_MakeEmptyRaster, ST_Tile, and ST_FromGDALRaster
Raster Accessors	ST_GeoReference, ST_Height, ST_IsEmpty, ST_MetaData, ST_NumBands, ST_PixelHeight, ST_PixelWidth, ST_ScaleX, ST_ScaleY, ST_RasterToWorldCoord, ST_RasterToWorldCoordX, ST_RasterToWorldCoordY, ST_Rotation, ST_SkewX, ST_SkewY, ST_SRID, ST_Summary, ST_UpperLeftX, ST_UpperLeftY, ST_Width, ST_WorldToRasterCoord, ST_WorldToRasterCoordX, ST_WorldToRasterCoordY
Raster Band Accessors	ST_BandMetaData, ST_BandNoDataValue, ST_BandIsNoData, ST_BandPath, ST_BandPixelType, and ST_HasNoBand
Raster Pixel Accessors and Setters	ST_PixelAsPolygon, ST_PixelAsPolygons, ST_PixelAsPoint, ST_PixelAsPoints, ST_PixelAsCentroid, ST_PixelAsCentroids, ST_Value, ST_NearestValue, ST_Neighborhood, ST_SetValue, ST_SetValues, ST_DumpValues, and ST_PixelOfValue
Raster Editors	ST_SetGeoReference, ST_SetRotation, ST_SetScale, ST_SetSkew, ST_SetSRID, ST_SetUpperLeft, ST_Resample, ST_Rescale, ST_Reskew, and ST_SnapToGrid, ST_Resize, and ST_Transform
Raster Band Editors	ST_SetBandNoDataValue and ST_SetBandIsNoData
Raster Band Statistics and Analytics	ST_Count, ST_CountAgg, ST_Histogram, ST_Quantile, ST_SummaryStats, ST_SummaryStatsAgg, and ST_ValueCount
Raster Outputs	ST_AsBinary, ST_AsGDALRaster, ST_AsJPEG, ST_AsPNG, and ST_AsTIFF

Category	Function
Raster Processing	ST_Clip, ST_ColorMap, ST_Intersection, ST_MapAlgebra, ST_Reclass, and ST_Union ST_Distinct4ma, ST_InvDistWeight4ma, ST_Max4ma, ST_Mean4ma, ST_Min4ma, ST_MinDist4ma, ST_Range4ma, ST_StdDev4ma, and ST_Sum4ma, ST_Aspect, ST_HillShade, ST_Roughness, ST_Slope, ST_TPI, ST_TRI, Box3D, ST_ConvexHull, ST_DumpAsPolygons, and ST_Envelope, ST_MinConvexHull, ST_Polygon, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Disjoint, ST_Intersects, and ST_Overlaps, ST_Touches, ST_SameAlignment, ST_NotSameAlignmentReason, ST_Within, ST_DWithin, and ST_DFullyWithin
Raster Operators	&&, &<, &>, =, @, ~=, and ~

Table 12-2 Operators and functions supported by PostGIS3.2.2

Category	Function
Management functions	AddGeometryColumn, DropGeometryColumn, DropGeometryTable, PostGIS_Full_Version, PostGIS_GEOS_Version, PostGIS_Liblwgeom_Version, PostGIS_Lib_Build_Date, PostGIS_Lib_Version, PostGIS_PROJ_Version, PostGIS_Scripts_Build_Date, PostGIS_Scripts_Installed, PostGIS_Version, PostGIS_LibXML_Version, PostGIS_Scripts_Released, Populate_Geometry_Columns, UpdateGeometrySRID, PostGIS_Libprotobuf_Version, PostGIS_Wagyu_Version
Geometry constructors	ST_BdPolyFromText, ST_BdMPolyFromText, ST_Box2dFromGeoHash, ST_GeneratePoints, ST_GeogFromText, ST_GeographyFromText, ST_GeogFromWKB, ST_GeomCollFromText, ST_GeomFromEWKB, ST_GeomFromEWKT, ST_GeometryFromText, ST_GeomFromGeoHash, ST_GeomFromGML, ST_GeomFromGeoJSON, ST_GeomFromKML, ST_GMLToSQL, ST_GeomFromText, ST_GeomFromWKB, ST_LineFromMultiPoint, ST_LineFromText, ST_LineFromWKB, ST_LinestringFromWKB, ST_MakeBox2D, ST_3DMakeBox, ST_MakeEnvelope, ST_MakePolygon, ST_MakePoint, ST_MakePointM, ST_MLineFromText, ST_MPointFromText, ST_MPolyFromText, ST_Point, ST_Points, ST_PointFromGeoHash, ST_PointFromText, ST_PointFromWKB, ST_Polygon, ST_PolygonFromText, ST_WKBToSQL, ST_WKTToSQL, Geography_Distance_Knn, Geometry_Distance_Cpa, Geometry_Hash, ST_3Dlineinterpolate, ST_AsEncodedPolyline

Category	Function
Geometry accessors	GeometryType, ST_Boundary, ST_CoordDim, ST_Dimension, ST_EndPoint, ST_Envelope, ST_ExteriorRing, ST_GeometryN, ST_GeometryType, ST_InteriorRingN, ST_IsClosed, ST_IsCollection, ST_IsEmpty, ST_IsPolygonCCW, ST_IsPolygonCW, ST_IsRing, ST_IsSimple, ST_IsValid, ST_IsValidReason, ST_IsValidDetail, ST_M, ST_NDims, ST_NPoints, ST_NRings, ST_NumGeometries, ST_NumInteriorRings, ST_NumInteriorRing, ST_NumPatches, ST_NumPoints, ST_PatchN, ST_PointN, ST_SRID, ST_StartPoint, ST_Summary, ST_X, ST_XMax, ST_XMin, ST_Y, ST_YMax, ST_YMin, ST_Z, ST_ZMax, ST_Zmflag, ST_ZMin, ST_Wrapx, ST_Asmvt
Geometry editors	ST_AddPoint, ST_Affine, ST_Force2D, ST_Force3D, ST_Force3DZ, ST_Force3DM, ST_Force4D, ST_ForceCollection, ST_ForcePolygonCCW, ST_ForcePolygonCW, ST_ForceSFS, ST_ForceRHR, ST_LineMerge, ST_CollectionExtract, ST_CollectionHomogenize, ST_Multi, ST_Normalize, ST_RemovePoint, ST_Reverse, ST_Rotate, ST_RotateX, ST_RotateY, ST_RotateZ, ST_Scale, ST_Segmentize, ST_SetPoint, ST_SetSRID, ST_SnapToGrid, ST_Snap, ST_Transform, ST_Translate, ST_TransScale, ST_AsmvtGeom, ST_isvalidTrajectory, ST_linefromencodedpolyline, ST_lineinterpolatepoints, ST_MaximumInScribedCircle, ST_OrientedEnvelope, ST_QuantizeCoordinates, ST_ReducePrecision, ST_Scroll, ST_SetEffectiveArea, ST_simplifyvw, ST_square, ST_squaregrid, ST_Swapordinates, ST_VoronoiLines, ST_VoronoiPolygons
Geometry outputs	ST_AsBinary, ST_AsEWKB, ST_AsEWKT, ST_AsGeoJSON, ST_AsGML, ST_AsHEXEWKB, ST_AsKML, ST_AsLatLonText, ST_AsSVG, ST_AsText, ST_AsTwkb, ST_AsX3D, ST_GeoHash, ST_Json, ST_Jsonb, ST_GeomfromGeojson
Operators	&& , &&& , &< , &< , &> , << , << , = , >> , @ , &> , >> , ~ , ~= , <-> , <#> , <-> , = , <<->>

Category	Function
Spatial relationships and measurements	ST_3DClosestPoint, ST_3DDistance, ST_3DDWithin, ST_3DDFullyWithin, ST_3DIntersects, ST_3DLongestLine, ST_3DMaxDistance, ST_3DShortestLine, ST_Area, ST_Azimuth, ST_Centroid, ST_ClosestPoint, ST_Contains, ST_ContainsProperly, ST_Covers, ST_CoveredBy, ST_Crosses, ST_LineCrossingDirection, ST_Disjoint, ST_Distance, ST_HausdorffDistance, ST_MaxDistance, ST_DistanceSphere, ST_DistanceSpheroid, ST_DFullyWithin, ST_DWithin, ST_Equals, ST_HasArc, ST_Intersects, ST_Length, ST_Length2D, ST_3DLength, ST_LengthSpheroid, ST_Length2DSpheroid, ST_LongestLine, ST_MinimumBoundingRadius, ST_OrderingEquals, ST_Overlaps, ST_Perimeter, ST_Perimeter2D, ST_3DPerimeter, ST_PointOnSurface, ST_Project, ST_Relate, ST_RelateMatch, ST_ShortestLine, ST_Touches, ST_Within, _ST_DistancerecTree, _ST_DistancerecTreeCached, _ST_SorTableHash
Geometry processing	ST_Buffer, ST_BuildArea, ST_ClipByBox2D, ST_ClusterDBSCAN, ST_ClusterIntersecting, ST_ClusterKMeans, ST_ClusterWithin, ST_Collect, ST_ConcaveHull, ST_ConvexHull, ST_CurveToLine, ST_DelaunayTriangles, ST_Difference, ST_Dump, ST_DumpPoints, ST_DumpRings, ST_FlipCoordinates, ST_Intersection, ST_LineToCurve, ST_MakeValid, ST_MemUnion, ST_MinimumBoundingCircle, ST_Polygonize, ST_Node, ST_OffsetCurve, ST_RemoveRepeatedPoints, ST_SharedPaths, ST_ShiftLongitude, ST_Simplify, ST_SimplifyPreserveTopology, ST_Split, ST_Subdivide, ST_SymDifference, ST_Union, ST_UnaryUnion, ST_BoundingDiagonal, ST_ChaikinsMoothing, ST_ClosestPointofApproach, ST_CollectionExtract, ST_CPAwithin, ST_DistanceCPA, ST_DumpSegments, ST_EstimatedExtent, ST_Filterbym, ST_SetEffectiveArea, ST_Forcecurve
Linear referencing	ST_LineInterpolatePoint, ST_LineLocatePoint, ST_LineSubstring, ST_LocateAlong, ST_LocateBetween, ST_LocateBetweenElevations, ST_InterpolatePoint, ST_AddMeasure
Miscellaneous functions	Array_Agg, Box2D, Box3D, ST_Expand, ST_Extent, ST_3Dextent, Find_SRID, ST_MemSize
Exceptional functions	PostGIS_AddBBox, PostGIS_DropBBox, PostGIS_HasBBox

Spatial Indexes

In DWS, PostGIS Extension supports Generalized Search Tree (GiST) spatial indexes. This index type is inapplicable to partitioned tables. Different from B-tree

indexes, GiST indexes are suitable for any type of unconventional data structure and can effectively improve the retrieval efficiency of geometric and geographic data.

Run the following command to create a GiST index:

```
CREATE INDEX indexname ON tablename USING gist ( geometryfield );
```

Extension Constraints

- Only row-store tables are supported. Column-store indexes are not supported.
- Only Oracle-compatible databases are supported.
- The topology object management module, Topology, is not supported.
- BRIN indexes are not supported.
- The **spatial_ref_sys** table can only be queried during scale-out.

Plug-in Upgrade Compatibility

When upgrading from PostGIS 2.4.2 to 3.2.2, note that certain functions may become incompatible or not fully forward compatible. This can lead to inconsistencies in the functionality before and after the upgrade. Therefore, it is necessary to assess the impact of upgrade incompatibility on your services.

The following table provides compatibility details for related functions.

Category	PostGIS 2.4.2	PostGIS 3.2.2
Added functions in 3.2.2	N/A	ST_IsPolygonCW(geometry)
	N/A	ST_IsPolygonCCW(geometry)
	N/A	ST_PointInsideCircle(geometry,float8,float8,float8)
	N/A	ST_ForcePolygonCW(geometry)
	N/A	ST_ForcePolygonCCW(geometry)
	N/A	ST_Normalize(geom geometry)
	N/A	ST_AsTWKB(geom geometry, prec int4 default 0, prec_z int4 default 0, prec_m int4 default 0, with_sizes boolean default false, with_boxes boolean default false)
	N/A	ST_AsTWKB(geom geometry[], ids bigint[], prec int4 default 0, prec_z int4 default 0, prec_m int4 default 0, with_sizes boolean default false, with_boxes boolean default false)
	N/A	ST_MakeLine (geometry[])

Category	PostGIS 2.4.2	PostGIS 3.2.2
	N/A	ST_TileEnvelope(zoom integer, x integer, y integer, bounds geometry DEFAULT 'SRID=3857;LINESTRING(-20037508.342789244 -20037508.342789244, 20037508.342789244 20037508.342789244)::geometry, margin float8 DEFAULT 0.0)
	N/A	ST_ClusterIntersecting(geometry[])
	N/A	ST_ClusterWithin(geometry[], float8)
	N/A	ST_ClusterDBSCAN (geometry, eps float8, minpoints int)
	N/A	ST_Scale(geometry, geometry, origin geometry)
	N/A	ST_GeneratePoints(area geometry, npoints integer, seed integer)
	N/A	ST_FrechetDistance(geom1 geometry, geom2 geometry, float8 default -1)
	N/A	ST_Points(geometry)
	N/A	ST_ClipByBox2d(geom geometry, box box2d)
	N/A	ST_Subdivide(geom geometry, maxvertices integer DEFAULT 256, gridSize float8 DEFAULT -1.0)
	N/A	ST_ClusterIntersecting (geometry)
	N/A	ST_ClusterWithin (geometry, float8)
	N/A	ST_ClusterKMeans(geom geometry, k integer, max_radius float8 default null)
	N/A	ST_AsText(geometry, int4)
	N/A	ST_AsEWKT(geography, int4)
	N/A	_ST_CoveredBy(geog1 geography, geog2 geography)

Category	PostGIS 2.4.2	PostGIS 3.2.2
	N/A	ST_Point(float8, float8, srid integer)
Functions no longer supported in 3.2.2	ST_3DLength_spheroid(geometry, spheroid)	N/A
	ST_length2d_spheroid(geometry, spheroid)	N/A
	ST_locate_between_measures(geometry, float8, float8)	N/A
	ST_locate_along_measure(geometry, float8)	N/A
	ST_Buffer(geometry, float8, text)	N/A
	ST_GeneratePoints(area geometry, npoints integer)	N/A
	ST_Combine_BBox(box3d, geometry)	N/A
	ST_Combine_BBox(box2d, geometry)	N/A
	pgis_abs_in(cstring)	N/A
	pgis_abs_out(pgis_abs)	N/A
	pgis_abs (internallength = 16, input = pgis_abs_in, output = pgis_abs_out, alignment = double)	N/A
	ST_MemUnion(geometry)	N/A
	pgis_geometry_accum_finalfn(pgis_abs)	N/A
	ST_MakeLine (geometry)	N/A
	ST_Accum (geometry)	When a single geometry data record is input, its value is output directly. However, if multiple geometry data records are provided, an error message will be displayed, indicating that multiple records are unsupported.
	_ST_AsKML(int4, geometry, int4, text)	N/A
	ST_MemUnion(geometry)	N/A

Category	PostGIS 2.4.2	PostGIS 3.2.2
	<code>_ST_AsGeoJson(int4, geometry, int4, int4)</code>	N/A
	<code>ST_AsGeoJson(gj_version int4, geom geometry, maxdecimaldigits int4 DEFAULT 15, options int4 DEFAULT 0)</code>	N/A
	<code>_ST_DWithin(geography, geography, float8, boolean)</code>	N/A
	<code>ST_point_inside_circle(geometry, float8, float8, float8)</code>	N/A
	<code>ST_CurveToLine(geometry)</code>	N/A
	<code>ST_Shift_Longitude(geometry)</code>	Use ST_ShiftLongitude instead.
	<code>ST_find_extent(text, text, text)</code> and <code>ST_find_extent(text, text)</code>	Use ST_FindExtent instead.
	<code>ST_mem_size(geometry)</code>	Use ST_MemSize instead.
	<code>ST_length_spheroid(geometry, spheroid)</code>	Use ST_LengthSpheroid instead.
	<code>ST_distance_spheroid(geom1 geometry, geom2 geometry, spheroid)</code>	Use ST_DistanceSpheroid instead.
	<code>ST_force_2d(geometry)</code>	Use ST_Force2D instead.
	<code>ST_force_3dz(geometry)</code>	Use ST_Force3DZ instead.
	<code>ST_force_3d(geometry)</code>	Use ST_Force3D instead.
	<code>ST_force_3dm(geometry)</code>	Use ST_Force3DM instead.
	<code>ST_force_4d(geometry)</code>	Use ST_Force4D instead.
	<code>ST_force_collection(geometry)</code>	Use ST_ForceCollection instead.
	<code>ST_line_locate_point(geom1 geometry, geom2 geometry)</code>	Use ST_LineLocatePoint instead.
	<code>ST_line_interpolate_point(geometry, float8)</code>	Use ST_LineInterpolatePoint instead.

Category	PostGIS 2.4.2	PostGIS 3.2.2
	ST_Buffer(geometry,float8)	Use ST_Buffer instead.
Functions with parameter type changed in 3.2.2	pgis_geometry_accum_transfn(pgis_abs, geometry)	pgis_geometry_accum_transfn(internal, geometry)
	pgis_geometry_accum_transfn(pgis_abs, geometry, float8)	pgis_geometry_accum_transfn(internal, geometry, float8)
	pgis_geometry_accum_transfn(pgis_abs, geometry, float8, int)	pgis_geometry_accum_transfn(internal, geometry, float8, int)
	pgis_geometry_union_finalfn(pgis_abs)	pgis_geometry_union_finalfn(internal)
	pgis_geometry_collect_finalfn(pgis_abs)	pgis_geometry_collect_finalfn(internal)
	pgis_geometry_polygonize_finalfn(pgis_abs)	pgis_geometry_polygonize_finalfn(internal)
	pgis_geometry_clusterintersecting_finalfn(pgis_abs)	pgis_geometry_clusterintersecting_finalfn(internal)
	ST_Union (geometry)	ST_Union (geometry)
	ST_Collect (geometry)	ST_Collect (geometry)
	ST_Buffer(geometry,float8, integer)	ST_Buffer(geom geometry, radius float8, quadsegs integer)
Functions with API changed in 3.2.2	ST_AsKML(version int4, geom geometry, maxdecimaldigits int4 DEFAULT 15, nprefix text DEFAULT null)	ST_AsKML(geom geometry, maxdecimaldigits int4 DEFAULT 15, nprefix TEXT default ' ')
	ST_AsKML(geom geometry, maxdecimaldigits int4 DEFAULT 15)	ST_AsKML(geom geometry, maxdecimaldigits int4 DEFAULT 15, nprefix TEXT default ' ')
	ST_AsKML(version int4, geom geometry, maxdecimaldigits int4 DEFAULT 15, nprefix text DEFAULT null)	ST_AsGML(version int4, geog geography, maxdecimaldigits int4 DEFAULT 15, options int4 DEFAULT 0, nprefix text DEFAULT 'gml', id text DEFAULT '')
Functions with default parameters changed in 3.2.2	ST_SymDifference(geom1 geometry, geom2 geometry)	ST_SymDifference(geom1 geometry, geom2 geometry, gridSize float8 DEFAULT -1.0)

Category	PostGIS 2.4.2	PostGIS 3.2.2
	ST_UnaryUnion(geometry)	ST_UnaryUnion(geometry, gridSize float8 DEFAULT -1.0)
	ST_AsGeoJson(geom geometry, maxdecimaldigits int4 DEFAULT 15, options int4 DEFAULT 0)	ST_AsGeoJson(geom geometry, maxdecimaldigits int4 DEFAULT 9, options int4 DEFAULT 8)
	ST_Buffer(geometry,float8,cstring)	ST_Buffer(geom geometry, radius float8, options text DEFAULT ' ')
	_ST_DWithin(geography, geography, float8, boolean)	_ST_DWithin(geog1 geography, geog2 geography, tolerance float8, use_spheroid boolean DEFAULT true)
	ST_IsValidDetail(geometry)	ST_IsValidDetail(geom geometry, flags int4 DEFAULT 0)
	ST_CurveToLine(geom geometry, tol float8, toltype integer, flags integer)	ST_CurveToLine(geom geometry, tol float8 DEFAULT 32, toltype integer DEFAULT 0, flags integer DEFAULT 0)
Functions supported hash join and merge join in 3.2.2	OPERATOR =	OPERATOR =
Function changes in version 3.2.2	ST_ConcaveHull weak verification	Strong verification is added to ST_ConcaveHull to ensure more rigorous verification.
Functions with commutor undefined in 3.2.2	OPERATOR &< ,OPERATOR &< ,OPERATOR &>	OPERATOR &< ,OPERATOR &< ,OPERATOR &>

12.4 OPEN SOURCE SOFTWARE NOTICE (For PostGIS)

This document contains open source software notice for the product. And this document is confidential information of copyright holder. Recipient shall protect it in due care and shall not disseminate it without permission.

Warranty Disclaimer

This document is provided "as is" without any warranty whatsoever, including the accuracy or comprehensiveness. Copyright holder of this document may change

the contents of this document at any time without prior notice, and copyright holder disclaims any liability in relation to recipient's use of this document.

Open source software is provided by the author "as is" and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of data or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of open source software, even if advised of the possibility of such damage.

Copyright Notice And License Texts

Software: postgis-2.4.2

Copyright notice:

"Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Copyright 2008 Kevin Neufeld

Copyright (c) 2009 Walter Bruce Sinclair

Copyright 2006-2013 Stephen Woodbridge.

Copyright (c) 2008 Walter Bruce Sinclair

Copyright (c) 2012 TJ Holowaychuk <tj@vision-media.ca>

Copyright (c) 2008, by Attractive Chaos <attractivechaos@aol.co.uk>

Copyright (c) 2001-2012 Walter Bruce Sinclair

Copyright (c) 2010 Walter Bruce Sinclair

Copyright 2006 Stephen Woodbridge

Copyright 2006-2010 Stephen Woodbridge.

Copyright (c) 2006-2014 Stephen Woodbridge.

Copyright (c) 2017, Even Rouault <even.rouault at spatialys.com>

Copyright (C) 2004-2015 Sandro Santilli <strk@kbt.io>

Copyright (C) 2008-2011 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2008 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>

Copyright 2015 Nicklas Avén <nicklas.aven@jordogskog.no>

Copyright 2008 Paul Ramsey

Copyright (C) 2012 Sandro Santilli <strk@kbt.io>

Copyright 2012 Sandro Santilli <strk@kbt.io>

Copyright (C) 2014 Sandro Santilli <strk@kbt.io>
Copyright 2013 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2009 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright 2008 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright 2011 Sandro Santilli <strk@kbt.io>
Copyright 2015 Daniel Baston
Copyright 2009 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2014 Kashif Rasul <kashif.rasul@gmail.com> and
Shoaib Burq <saburq@gmail.com>
Copyright 2013 Sandro Santilli <strk@kbt.io>
Copyright 2010 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2017 Sandro Santilli <strk@kbt.io>
Copyright (C) 2015 Sandro Santilli <strk@kbt.io>
Copyright (C) 2009 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2011 Sandro Santilli <strk@kbt.io>
Copyright 2010 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2014 Nicklas Avén
Copyright 2011-2016 Regina Obe
Copyright (C) 2008 Paul Ramsey
Copyright (C) 2011-2015 Sandro Santilli <strk@kbt.io>
Copyright 2010-2012 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright (C) 2015 Daniel Baston <dbaston@gmail.com>
Copyright (C) 2013 Nicklas Avén
Copyright (C) 2016 Sandro Santilli <strk@kbt.io>
Copyright 2017 Darafei Praliaskouski <me@komzpa.net>
Copyright (c) 2016, Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2011-2012 Sandro Santilli <strk@kbt.io>
Copyright (C) 2011 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2007-2008 Mark Cave-Ayland
Copyright (C) 2001-2006 Refrations Research Inc.
Copyright 2015 Daniel Baston <dbaston@gmail.com>
Copyright 2009 David Skea <David.Skea@gov.bc.ca>
Copyright (C) 2012-2015 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2012-2015 Sandro Santilli <strk@kbt.io>
Copyright 2001-2006 Refrations Research Inc.
Copyright (C) 2004 Refrations Research Inc.
Copyright 2011-2014 Sandro Santilli <strk@kbt.io>
Copyright 2009-2010 Sandro Santilli <strk@kbt.io>
Copyright 2015-2016 Daniel Baston <dbaston@gmail.com>
Copyright 2011-2015 Sandro Santilli <strk@kbt.io>
Copyright 2007-2008 Mark Cave-Ayland
Copyright 2012-2013 Omlandia <infos@oslandia.com>
Copyright (C) 2015-2017 Sandro Santilli <strk@kbt.io>
Copyright (C) 2001-2003 Refrations Research Inc.
Copyright 2016 Sandro Santilli <strk@kbt.io>
Copyright 2011 Kashif Rasul <kashif.rasul@gmail.com>
Copyright (C) 2014 Nicklas Avén
Copyright (C) 2010 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2010-2015 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2011 Sandro Santilli <strk@kbt.io>
Copyright (C) 2011-2014 Sandro Santilli <strk@kbt.io>
Copyright (C) 1984, 1989-1990, 2000-2015 Free Software Foundation, Inc.
Copyright (C) 2011 Paul Ramsey
Copyright 2001-2003 Refrations Research Inc.
Copyright 2009-2010 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright 2010-2012 Omlandia
Copyright 2006 Corporacion Autonoma Regional de Santander
Copyright 2013 Nicklas Avén
Copyright 2011-2016 Arrival 3D, Regina Obe
Copyright (C) 2009 David Skea <David.Skea@gov.bc.ca>
Copyright (C) 2017 Sandro Santilli <strk@kbt.io>
Copyright (C) 2009-2012 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2010 - Omlandia
Copyright (C) 2006 Mark Leslie <mark.leslie@lisoftware.com>
Copyright (C) 2008-2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (C) 2009-2015 Paul Ramsey <pramsey@cleverelephant.ca>

Copyright (C) 2010 Olivier Courtin <olivier.courtin@camptocamp.com>
Copyright 2010 Nicklas Avén
Copyright 2012 Paul Ramsey
Copyright 2011 Nicklas Avén
Copyright 2002 Thamer Alharbash
Copyright 2011 OSGeo
Copyright (C) 2009-2011 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2008 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (C) 2004-2007 Refrations Research Inc.
Copyright 2010 LISAssoft Pty Ltd
Copyright 2010 Mark Leslie
Copyright (c) 1999, Frank Warmerdam
Copyright 2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (c) 2007, Frank Warmerdam
Copyright 2008 OpenGeo.org
Copyright (C) 2008 OpenGeo.org
Copyright (C) 2009 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright 2010 LISAssoft
Copyright (C) 2010 Mark Cave-Ayland <mark.cave-ayland@siriusit.co.uk>
Copyright (c) 1999, 2001, Frank Warmerdam
Copyright (C) 2016-2017 Bjørn Harrell <bjorn@wololo.org>
Copyright (C) 2017 Danny Goette <danny.goette@fem.tu-ilmenau.de>
Copyright 2009-2011 Paul Ramsey <pramsey@cleverelephant.ca>
^copyright^
Copyright 2012 (C) Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2006 Refrations Research Inc.
Copyright 2009 Paul Ramsey <pramsey@opengeo.org>
Copyright 2001-2009 Refrations Research Inc.
Copyright (C) 2010 Olivier Courtin <olivier.courtin@oslandia.com>
By Nathan Wagner, copyright disclaimed,
this entire file is in the public domain
Copyright 2009-2011 Olivier Courtin <olivier.courtin@oslandia.com>
Copyright (C) 2001-2005 Refrations Research Inc.

Copyright 2001-2011 Refrations Research Inc.
Copyright 2009-2014 Sandro Santilli <strk@kbt.io>
Copyright (C) 2008 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2010 Sandro Santilli <strk@kbt.io>
Copyright 2012 J Smith <dark.panda@gmail.com>
Copyright 2009 - 2010 Oslandia
Copyright 2009 Oslandia
Copyright 2001-2005 Refrations Research Inc.
Copyright 2016 Paul Ramsey <pramsey@cleverelephant.ca>
Copyright 2016 Daniel Baston <dbaston@gmail.com>
Copyright (C) 2011 OpenGeo.org
Copyright (c) 2003-2017, Troy D. Hanson <http://troydhanson.github.com/uthash/>
Copyright (C) 2011 Regents of the University of California
Copyright (C) 2011-2013 Regents of the University of California
Copyright (C) 2010-2011 Jorge Arevalo <jorge.arevalo@deimos-space.com>
Copyright (C) 2010-2011 David Zwarg <dzwarg@azavea.com>
Copyright (C) 2009-2011 Pierre Racine <pierre.racine@sbflaval.ca>
Copyright (C) 2009-2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2008-2009 Sandro Santilli <strk@kbt.io>
Copyright (C) 2013 Nathaniel Hunter Clay <clay.nathaniel@gmail.com>
Copyright (C) 2013 Nathaniel Hunter Clay <clay.nathaniel@gmail.com>
Copyright (C) 2013 Bborie Park <dustymugs@gmail.com>
Copyright (C) 2013 Nathaniel Hunter Clay <clay.nathaniel@gmail.com>
(C) 2009 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2009 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2009-2010 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2009-2010 Jorge Arevalo <jorge.arevalo@deimos-space.com>
Copyright (C) 2012 Regents of the University of California
Copyright (C) 2013 Regents of the University of California
Copyright (C) 2012-2013 Regents of the University of California
Copyright (C) 2009 Sandro Santilli <strk@kbt.io>
"

License: The GPL v2 License.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.

51 Franklin St, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain

patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.?

GNU GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a

warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER

PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author

Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.

This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs. If your program is a subroutine library, you may consider it
more useful to permit linking proprietary applications with the library. If this is
what you want to do, use the GNU Library General Public License instead of this
License.

Software:Geos

Copyright notice:

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2013 Sandro Santilli <strk@keybit.net>

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frii.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frii.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frii.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refrations Research

Copyright 2004 Sean Gillies, sgillies@frie.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Niall Dawson <niall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)
Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE
Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth
Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license
document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.
Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2005 2006 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.
Copyright (C) 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2016 Daniel Baston
Copyright (C) 2008 Sean Gillies
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Refrations Research Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Sandro Santilli <strk@keybit.net>
Copyright (C) 2008-2010 Safe Software Inc.
Copyright (C) 2006-2007 Refrations Research Inc.
Copyright (C) 2005-2007 Refrations Research Inc.
Copyright (C) 2007 Refrations Research Inc.
Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>
Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>
Copyright (C) 2010 Sandro Santilli <strk@keybit.net>
Copyright (C) 2009 Mateusz Loskot
Copyright (C) 2005-2009 Refrations Research Inc.
Copyright (C) 2001-2009 Vivid Solutions Inc.
Copyright (C) 2012 Sandro Santilli <strk@keybit.net>
Copyright (C) 2006 Wu Yongwei
Copyright (C) 2012 Excensus LLC.
Copyright (C) 1996-2015 Free Software Foundation, Inc.
Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>
Copyright (C) 2007-2010 Safe Software Inc.
Copyright (C) 2010 Safe Software Inc.
Copyright (C) 2006 Refrations Research
Copyright 2004 Sean Gillies, sgillies@frie.com
Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>
Copyright (C) 2015 Nyal Dawson <nyall dot dawson at gmail dot com>
Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Copyright (C) 2005-2011 Refrations Research Inc.

Copyright (C) 2009 Ragi Y. Burhum <ragi@burhum.com>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2005 2006 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006-2011 Refrations Research Inc.

Copyright (C) 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009-2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2016 Daniel Baston

Copyright (C) 2008 Sean Gillies

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Refrations Research Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Sandro Santilli <strk@keybit.net>

Copyright (C) 2008-2010 Safe Software Inc.

Copyright (C) 2006-2007 Refrations Research Inc.

Copyright (C) 2005-2007 Refrations Research Inc.

Copyright (C) 2007 Refrations Research Inc.

Copyright (C) 2014 Mika Heiskanen <mika.heiskanen@fmi.fi>

Copyright (C) 2009-2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 2011 Sandro Santilli <strk@keybit.net>

Copyright (C) 2010 Sandro Santilli <strk@keybit.net>

Copyright (C) 2009 Mateusz Loskot

Copyright (C) 2005-2009 Refrations Research Inc.

Copyright (C) 2001-2009 Vivid Solutions Inc.

Copyright (C) 2012 Sandro Santilli <strk@keybit.net>

Copyright (C) 2006 Wu Yongwei

Copyright (C) 2012 Excensus LLC.

Copyright (C) 1996-2015 Free Software Foundation, Inc.

Copyright (c) 1995 Olivier Devillers <Olivier.Devillers@sophia.inria.fr>

Copyright (C) 2007-2010 Safe Software Inc.

Copyright (C) 2010 Safe Software Inc.

Copyright (C) 2006 Refrations Research

Copyright 2004 Sean Gillies, sgillies@frie.com

Copyright (C) 2011 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2015 Nyall Dawson <nyall dot dawson at gmail dot com>

Original code (2.0 and earlier)copyright (c) 2000-2006 Lee Thomason
(www.grinninglizard.com)

Original code (2.0 and earlier)copyright (c) 2000-2002 Lee Thomason
(www.grinninglizard.com)

License: LGPL V2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1.]

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public

Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you

first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and

is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in

non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The

former contains code derived from the library, whereas the latter must be combined with the library in order to run.

GNU LESSER GENERAL PUBLIC LICENSE

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under

copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another

language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an

appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the

Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1

above, provided that you also meet all of these conditions:

a) The modified work must itself be a software library.

b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of

its purpose remains meaningful.

(For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d

requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in

themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or

collective works based on the Library.

In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany

it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to

distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a

work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License.

Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object

file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6,

whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work

under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work

(which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)

b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system,

rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception,

the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on

which the executable runs, unless that component itself accompanies the executable.

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot

use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein.

You are not responsible for enforcing compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the

integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide

range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time.

Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is

copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status

of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the library's name and a brief idea of what it does.>

Copyright (C) <year> <name of author>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU

Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library `Frob' (a library for tweaking knobs) written by James Random Hacker.

<signature of Ty Coon>, 1 April 1990

Ty Coon, President of Vice

That's all there is to it!

Software: JSON-C

Copyright notice:

Copyright (c) 2004, 2005 Metaparadigm Pte. Ltd.

Copyright (c) 2009-2012 Eric Haszlakiewicz

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd

Copyright (c) 2009 Hewlett-Packard Development Company, L.P.

Copyright 2011, John Resig

Copyright 2011, The Dojo Foundation

Copyright (c) 2012 Eric Haszlakiewicz

Copyright (c) 2009-2012 Hewlett-Packard Development Company, L.P.

Copyright (c) 2008-2009 Yahoo! Inc. All rights reserved.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,
2007, 2008, 2009, 2010, 2011 Free Software Foundation, Inc.

Copyright (c) 2013 Metaparadigm Pte. Ltd.

License: MIT License

Copyright (c) 2009-2012 Eric Haszlakiewicz

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Copyright (c) 2004, 2005 Metaparadigm Pte Ltd

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Software: proj

Copyright notice:

"Copyright (C) 2010 Mateusz Loskot <mateusz@loskot.net>

Copyright (C) 2007 Douglas Gregor <doug.gregor@gmail.com>

Copyright (C) 2007 Troy Straszheim

CMake, Copyright (C) 2009-2010 Mateusz Loskot <mateusz@loskot.net>)

Copyright (C) 2011 Nicolas David <nicolas.david@ign.fr>

Copyright (c) 2000, Frank Warmerdam

Copyright (c) 2011, Open Geospatial Consortium, Inc.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,
2007, 2008, 2009, 2010, 2011 Free Software Foundation, Inc.

Copyright (c) Charles Karney (2012-2015) <charles@karney.com> and licensed

Copyright (c) 2005, Antonello Andrea

Copyright (c) 2010, Frank Warmerdam

Copyright (c) 1995, Gerald Evenden

Copyright (c) 2000, Frank Warmerdam <warmerdam@pobox.com>

Copyright (c) 2010, Frank Warmerdam <warmerdam@pobox.com>

Copyright (c) 2013, Frank Warmerdam

Copyright (c) 2003 Gerald I. Evenden
Copyright (c) 2012, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2002, Frank Warmerdam
Copyright (c) 2004 Gerald I. Evenden
Copyright (c) 2012 Martin Raspaud
Copyright (c) 2001, Thomas Flemming, tf@ttqv.com
Copyright (c) 2002, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2009, Frank Warmerdam
Copyright (c) 2003, 2006 Gerald I. Evenden
Copyright (c) 2011, 2012 Martin Lambers <marlam@marlam.de>
Copyright (c) 2006, Andrey Kiselev
Copyright (c) 2008-2012, Even Rouault <even dot rouault at mines-paris dot org>
Copyright (c) 2001, Frank Warmerdam
Copyright (c) 2001, Frank Warmerdam <warmerdam@pobox.com>
Copyright (c) 2008 Gerald I. Evenden
"

License: MIT License

Please see above

Software: libxml2

Copyright notice:

"See Copyright for the status of this software.

Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.

Copyright (C) 2003 Daniel Veillard.

copy: see Copyright for the status of this software.

copy: see Copyright for the status of this software

copy: see Copyright for the status of this software.

Copyright (C) 2000 Bjorn Reese and Daniel Veillard.

Copy: See Copyright for the status of this software.

See COPYRIGHT for the status of this software

Copyright (C) 2000 Gary Pennington and Daniel Veillard.

Copyright (C) 1996, 1997, 1998, 1999, 2000, 2001, 2003, 2004, 2005, 2006,

2007 Free Software Foundation, Inc.

Copyright (C) 1998 Bjorn Reese and Daniel Stenberg.

Copyright (C) 2001 Bjorn Reese <breese@users.sourceforge.net>

Copyright (C) 2000 Bjorn Reese and Daniel Stenberg.

Copyright (C) 2001 Bjorn Reese and Daniel Stenberg.

See Copyright for the status of this software

"

License: MIT License

Please see above

13 Using JDBC or ODBC for DWS Secondary Development

13.1 Prerequisites

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

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

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

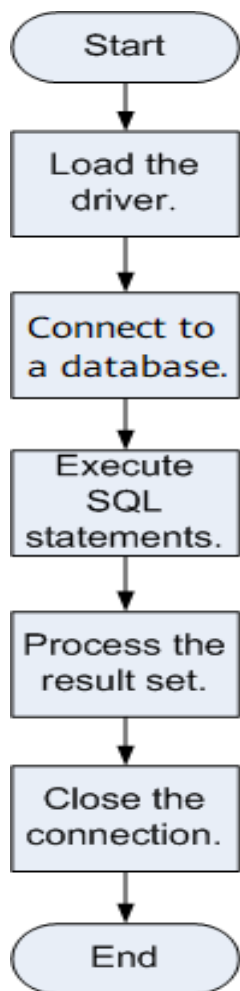
Downloading Drivers

For details, see "Downloading the JDBC or ODBC Driver" in the *Data Warehouse Service (DWS) User Guide*.

13.2 JDBC-Based Development

13.2.1 JDBC Development Process

Java Database Connectivity (JDBC) is a Java API for executing SQL statements. It provides a unified access interface for multiple relational databases, enabling applications to work with data based on it. DWS supports JDBC 4.0 and requires JDK 1.6 or later for code compiling. It does not support JDBC-ODBC Bridge. The following figure shows the JDBC application development process.

Figure 13-1 JDBC-based application development process**Table 13-1** JDBC development process

Procedure	Description
Load the driver.	Download the JDBC driver and edit and load it in the program.
Connect to a database.	Connect to the database through the JDBC driver.
Execute SQL statements.	Applications operate database data by executing SQL statements.
Process the result set.	Different types of result sets have different application scenarios. Applications need to select the appropriate result set type as needed.
Close the connection.	Make sure to close the database connection after completing the required data operations.

13.2.2 JDBC Package and Driver Class

JDBC Package

Download the **dws_9.1.x_jdbc_driver.zip** software package from the management console.

For details, see "Downloading the JDBC or ODBC Driver" in the *Data Warehouse Service (DWS) User Guide*.

JDBC driver JAR package obtained from decompression:

gsjdbc4.jar: Driver package compatible with PostgreSQL. The class name and class structure in the driver are the same as those in the PostgreSQL driver. All the applications running on PostgreSQL can be smoothly transferred to the current system.

Driver Class

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

13.2.3 Loading a Driver

Load the database driver before creating a database connection.

You can load the driver in the following ways:

- Implicitly loading the driver before creating a connection in the code:
Class.forName ("org.postgresql.Driver")
- Transferring a parameter during the JVM startup: **java -Djdbc.drivers=org.postgresql.Driver jdbctest**

13.2.4 Connecting to a Database

After a database is connected, you can run SQL statements the database to perform operations on data.

Prerequisites

If you use an open source JDBC driver, set **password_encryption_type** to **1**. If it is not **1**, the connection might fail with an error saying "none of the server's SASL authentication mechanisms are supported". To fix this, perform the following steps:

1. Check and change the **password_encryption_type** to **1** if needed with the help of technical support.
2. Create a new database user for connection or reset the password of the existing database user.
 - If you are an administrator, reset the password by referring to "Resetting the Password of the DWS Database Administrator" in the *Data Warehouse Service (DWS) User Guide*.

- If you are a common user, use another client tool (such as Data Studio) to connect to the database and run the **ALTER USER** statement to change your password.
3. Connect to the database.

Here are the reasons why you need to perform these operations:

- MD5 algorithms may be vulnerable to collision attacks and cannot be used for password verification. DWS uses the default security design. By default, MD5 password verification is disabled. As a result, open-source clients may fail to be connected. Therefore, you need to adjust the cryptographic algorithm parameter **password_encryption_type** to enable the MD5 algorithm.
- The database stores the hash digest (SHA-256 digest by default) of passwords instead of password text. During password verification, the system compares the hash digest with the password digest sent from the client (salt operations are involved). If you change your cryptographic algorithm policy, the database cannot generate a new MD5 hash digest for your existing password. For connectivity purposes, you must manually change your password or create a new user. The new password will be encrypted using the hash algorithm and stored for authentication in the next connection.

Function Prototype

JDBC provides the following three database connection methods:

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

Parameter

Table 13-2 Database connection parameters

Parameter	Description
url	gsjdbc4.jar database connection descriptor. The descriptor format can be: • jdbc:postgresql:database • jdbc:postgresql://host/database • jdbc:postgresql://host:port/database • jdbc:postgresql://host:port[,host:port][...]/database NOTE If gsjdbc200.jar is used, replace jdbc:postgresql with jdbc:gaussdb. • database: indicates the name of the database to be connected. • host indicates the name or IP address of the database server. If an ELB is bound to the cluster, set host to the IP address of the ELB. For security purposes, the CN forbids access from other nodes in the cluster without authentication. To access the CN from inside the cluster, deploy the JDBC program on the host where the CN is located and set host to 127.0.0.1. If you do not do so, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. It is recommended that the service system be deployed outside the cluster. If it is deployed inside, the database performance may be affected. • port: indicates the port number of a database server. By default, the database on port 8000 of the local host is connected. • Multiple IP addresses and ports can be configured. JDBC balances load by random access and failover, and will automatically ignore unreachable IP addresses. IP addresses are separated using commas. Example: jdbc:postgresql://10.10.0.13:8000,10.10.0.14:8000/database • If JDBC is used to connect to a cluster, only JDBC connection parameters can be configured in a cluster address. Variables cannot be added.

Parameter	Description
info	Database connection properties. Common properties include: • user: string type. It indicates the database user establishing a connection. • password: string type. It indicates the password of a database user. • ssl: Boolean type. It indicates whether the Secure Socket Layer (SSL) is used. • loggerLevel: string type. It indicates the amount of information that the driver logs and prints to the LogStream or LogWriter specified in the DriverManager. Currently, OFF, DEBUG, and TRACE are supported. DEBUG indicates that only logs of the DEBUG or higher level are printed, generating a few log information. TRACE indicates that logs of the DEBUG and TRACE levels are printed, generating detailed log information. The default value is OFF, indicating that no information will be logged. • prepareThreshold: integer type. It indicates the number of PreparedStatement executions required before SQL statements are switched over to servers as prepared statements. The default value is 5. • batchMode: boolean type. It indicates whether to connect the database in batch mode. • fetchsize: integer type. It indicates the default fetchsize for statements in the created connection. • ApplicationName: string type. It indicates an application name. The default value is PostgreSQL JDBC Driver. • allowReadOnly: boolean type. It indicates whether to enable the read-only mode for connection. The default value is false. If the value is not changed to true, the execution of connection.setReadOnly does not take effect. • blobMode: string type. It is used to set the setBinaryStream method to assign values to different data types. The value on indicates that values are assigned to the BLOB data type and off indicates that values are assigned to the bytea data type. The default value is on. • connectionExtraInfo: boolean type. It indicates whether the JDBC driver reports the driver deployment path and process owner to the database. NOTE The value can be true or false. The default value is true. If connectionExtraInfo is set to true, the JDBC driver reports the driver deployment path and process owner to the database and displays the information in the connection_info parameter (see connection_info). In this case, you can query the information from PG_STAT_ACTIVITY or PGXC_STAT_ACTIVITY.
user	Indicates a database user.

Parameter	Description
password	Indicates the password of a database user.

Closing the Connection

Make sure to close the database connection after completing the required data operations.

To close the database connection, you can directly invoke the **close** method, for example, **conn.close()**.

Examples

```
//gsjdbc4.jar is used as an example.
//The following code encapsulates database connection operations into an interface. The database can then
//be connected using an authorized username and password.

public static Connection GetConnection(String username, String passwd) {
    //Set the driver class.
    String driver = "org.postgresql.Driver";
    //Database connection descriptor.
    String sourceURL = "jdbc:postgresql://10.10.0.13:8000/postgres?currentSchema=test";
    Connection conn = null;

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

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

    return conn;
}
```

13.2.5 Executing SQL Statements

Executing an Ordinary SQL Statement

The application performs data (parameter statements do not need to be transferred) in the database by running SQL statements, and you need to perform the following steps:

Step 1 Create a statement object by triggering the `createStatement` method in `Connection`.

```
Statement stmt = con.createStatement();
```

Step 2 Execute the SQL statement by triggering the executeUpdate method in Statement.

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

 NOTE

If an execution request (not in a transaction block) received in the database contains multiple statements, the request is packed into a transaction. **VACUUM** is not supported in a transaction block. If one of the statements fails, the entire request will be rolled back.

Step 3 Close the statement object.

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

----End

Executing a Prepared SQL Statement

Pre-compiled statements were once compiled and optimized and can have additional parameters for different usage. For the statements have been pre-compiled, the execution efficiency is greatly improved. If you want to execute a statement for several times, use a precompiled statement. Perform the following procedure:

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

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

Step 2 Set parameters by triggering the setShort method in PreparedStatement.

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

Step 3 Execute the precompiled SQL statement by triggering the executeUpdate method in PreparedStatement.

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

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

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

----End

Calling a Stored Procedure

Perform the following steps to call existing stored procedures through the JDBC interface in DWS:

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

```
CallableStatement cstmt = myConn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
```

Step 2 Set parameters by calling the setInt method in CallableStatement.

```
cstmt.setInt(2, 50);  
cstmt.setInt(1, 20);  
cstmt.setInt(3, 90);
```

Step 3 Register with an output parameter by calling the registerOutParameter method in CallableStatement.

```
cstmt.registerOutParameter(4, Types.INTEGER); //Register an OUT parameter as an integer.
```

Step 4 Call the stored procedure by calling the execute method in CallableStatement.

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

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

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

For example:

```
//The following stored procedure has been created with the OUT parameter:  
create or replace procedure testproc  
(  
    psv_in1 in integer,  
    psv_in2 in integer,  
    psv_inout in out integer  
)  
as  
begin  
    psv_inout := psv_in1 + psv_in2 + psv_inout;  
end;  
/
```

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

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

NOTE

- Many database classes such as Connection, Statement, and ResultSet have a close() method. Close these classes after using their objects. Closing Connection will close all the related Statements, and closing a Statement will close its ResultSet.
- Some JDBC drivers support named parameters, which can be used to set parameters by name rather than sequence. If a parameter has a default value, you do not need to specify any parameter value but can use the default value directly. Even though the parameter sequence changes during a stored procedure, the application does not need to be modified. Currently, the DWS JDBC driver does not support this method.
- DWS does not support functions containing OUT parameters, or default values of stored procedures and function parameters.

----End

NOTICE

- If JDBC is used to call a stored procedure whose returned value is a cursor, the returned cursor cannot be used.
 - A stored procedure and an SQL statement must be executed separately.
-

Batch Processing

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

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

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO customer_t1 VALUES (?)");
```

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

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

Step 3 Execute batch processing by calling the `executeBatch` method in `PreparedStatement`.

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

Step 4 Close the precompiled statement object by calling the `close` method in `PreparedStatement`.

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

NOTE

Do not terminate a batch processing action when it is ongoing; otherwise, the database performance will deteriorate. Therefore, disable the automatic submission function during batch processing, and manually submit every several lines. The statement for disabling automatic submission is **`conn.setAutoCommit(false)`**.

----End

13.2.6 Processing Data in a Result Set

Setting a Result Set Type

Different types of result sets are applicable to different application scenarios. Applications select proper types of result sets based on requirements. Before executing an SQL statement, you must create a statement object. Some methods of creating statement objects can set the type of a result set. [Table 13-3](#) lists result set parameters. The connection methods are as follows:

```
//Create a Statement object. This object will generate a ResultSet object with a specified type and  
concurrency.  
createStatement(int resultSetType, int resultSetConcurrency);
```

```
//Create a PreparedStatement object. This object will generate a ResultSet object with a specified type and  
concurrency.  
prepareStatement(String sql, int resultSetType, int resultSetConcurrency);
```

```
//Create a CallableStatement object. This object will generate a ResultSet object with a specified type and  
concurrency.  
prepareCall(String sql, int resultSetType, int resultSetConcurrency);
```

Table 13-3 Result set types

Parameter	Description
resultSetType	Indicates the type of a result set. There are three types of result sets: • ResultSet.TYPE_FORWARD_ONLY: The ResultSet object can only be navigated forward. It is the default value. • ResultSet.TYPE_SCROLL_SENSITIVE: You can view the modified result by scrolling to the modified row. • ResultSet.TYPE_SCROLL_INSENSITIVE: The ResultSet object is insensitive to changes in the underlying data source. NOTE After a result set has obtained data from the database, the result set is insensitive to data changes made by other transactions, even if the result set type is ResultSet.TYPE_SCROLL_SENSITIVE. To obtain up-to-date data of the record pointed by the cursor from the database, call the refreshRow() method in a ResultSet object.
resultSetConcurrency	Indicates the concurrency type of a result set. There are two types of concurrency. • ResultSet.CONCUR_READ_ONLY: The data in a result set cannot be updated except that an updated statement has been created in the result set data. • ResultSet.CONCUR_UPDATEABLE: changeable result set. The concurrency type for a result set object can be updated if the result set is scrollable.

Positioning a Cursor in a Result Set

ResultSet objects include a cursor pointing to the current data row. The cursor is initially positioned before the first row. The next method moves the cursor to the next row from its current position. When a **ResultSet** object does not have a next row, a call to the next method returns **false**. Therefore, this method is used in the while loop for result set iteration. However, the JDBC driver provides more cursor positioning methods for scrollable result sets, which allows positioning cursor in the specified row. [Table 13-4](#) lists these methods.

Table 13-4 Methods for positioning a cursor in a result set

Method	Description
next()	Moves cursor to the next row from its current position.
previous()	Moves cursor to the previous row from its current position.

Method	Description
beforeFirst()	Places cursor before the first row.
afterLast()	Places cursor after the last row.
first()	Places cursor to the first row.
last()	Places cursor to the last row.
absolute(int)	Places cursor to a specified row.
relative(int)	Moves cursor forward or backward a specified number of rows.

Obtaining the Cursor Position from a Result Set

This cursor positioning method will be used to change the cursor position for a scrollable result set. JDBC driver provides a method to obtain the cursor position in a result set. [Table 13-5](#) lists the method.

Table 13-5 Method for obtaining the cursor position in a result set

Method	Description
isFirst()	Checks whether the cursor is in the first row.
isLast()	Checks whether the cursor is in the last row.
isBeforeFirst()	Checks whether the cursor is before the first row.
isAfterLast()	Checks whether the cursor is after the last row.
getRow()	Gets the current row number of the cursor.

Obtaining Data in a Result Set

ResultSet objects provide a variety of methods to obtain data from a result set. [Table 13-6](#) lists the common methods for obtaining data. If you want to know more about other methods, see JDK official documents.

Table 13-6 Common methods for obtaining data from a result set

Method	Description
int getInt(int columnIndex)	Retrieves data of the int type by column.

Method	Description
int getInt(String columnLabel)	Retrieves data of the int type by column name.
String getString(int columnIndex)	Retrieves data of the string type by column.
String getString(String columnLabel)	Retrieves data of the string type by column name.
Date getDate(int columnIndex)	Retrieves data of the date type by column.
Date getDate(String columnLabel)	Retrieves data of the date type by column name.

13.2.7 Common JDBC Development Examples

Example 1

Before completing the following example, you need to create a stored procedure.

```
create or replace procedure testproc
(
    psv_in1 in integer,
    psv_in2 in integer,
    psv_inout in out integer
)
as
begin
    psv_inout := psv_in1 + psv_in2 + psv_inout;
end;
/
```

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

```
//DBtest.java
//gsjdbc4.jar is used as an example.
// This example illustrates the main processes of JDBC-based development, covering database connection
creation, table creation, and data insertion.

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

public class DBTest {

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

```
e.printStackTrace();
return null;
}

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

return conn;
};

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

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

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

//Run the preprocessing statement to insert data in batches.
public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
        //Generate a prepared statement.
        pst = conn.prepareStatement("INSERT INTO customer_t1 VALUES (?,?)");
        for (int i = 0; i < 3; i++) {
            //Add parameters.
            pst.setInt(1, i);
            pst.setString(2, "data " + i);
            pst.addBatch();
        }
        //Run batch processing.
        pst.executeBatch();
        pst.close();
    } catch (SQLException e) {
        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

//Run the precompilation statement to update data.
```

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

//Run a stored procedure.
public static void ExecCallableSQL(Connection conn) {
    CallableStatement cstmt = null;
    try {

        cstmt=conn.prepareCall("{? = CALL TESTPROC(?,?,?)}");
        cstmt.setInt(2, 50);
        cstmt.setInt(1, 20);
        cstmt.setInt(3, 90);
        cstmt.registerOutParameter(4, Types.INTEGER); //Register an OUT parameter as an integer.
        cstmt.execute();
        int out = cstmt.getInt(4); //Obtain the out parameter value.
        System.out.println("The CallableStatment TESTPROC returns:"+out);
        cstmt.close();
    } catch (SQLException e) {
        if (cstmt != null) {
            try {
                cstmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

/**
 * Main process. Call static methods one by one.
 * @param args
 */
public static void main(String[] args) {
    //Establish a connection to the database.
    Connection conn = GetConnection("tester", "password");

    //Create a table.
    CreateTable(conn);

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

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

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

    //Close the connection to the database.
```

```
try {  
    conn.close();  
} catch (SQLException e) {  
    e.printStackTrace();  
}  
  
}  
  
}
```

Example 2: High Client Memory Usage

In this example, **setFetchSize** adjusts the memory usage of the client by using the database cursor to obtain server data in batches. It may increase network interaction and damage some performance.

The cursor is valid within a transaction. Therefore, you need to disable the autocommit function.

```
// Disable the autocommit function.  
conn.setAutoCommit(false);  
Statement st = conn.createStatement();  
  
// Open the cursor and obtain 50 lines of data each time.  
st.setFetchSize(50);  
ResultSet rs = st.executeQuery("SELECT * FROM mytable");  
while (rs.next()){  
    System.out.print("a row was returned.");  
}  
rs.close();  
  
// Disable the server cursor.  
st.setFetchSize(0);  
rs = st.executeQuery("SELECT * FROM mytable");  
while (rs.next()){  
    System.out.print("many rows were returned.");  
}  
rs.close();  
  
// Close the statement.  
st.close();
```

Retrying SQL Queries for Applications

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

gsjdbc4.jar is used as an example.

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

```
*/
class ExitHandler extends Thread {
    private Statement cancel_stmt = null;

    public ExitHandler(Statement stmt) {
        super("Exit Handler");
        this.cancel_stmt = stmt;
    }
    public void run() {
        System.out.println("exit handle");
        try {
            this.cancel_stmt.cancel();
        } catch (SQLException e) {
            System.out.println("cancel query failed.");
            e.printStackTrace();
        }
    }
}

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

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

        return conn;
    }
}
```

Run an ordinary SQL statement. Create the **jdbc_test1** table.

```
public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();

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

        //Run an ordinary SQL statement.
        int rc2 = stmt
            .executeUpdate("DROP TABLE if exists jdbc_test1;");

        int rc1 = stmt
            .executeUpdate("CREATE TABLE jdbc_test1(col1 INTEGER, col2 VARCHAR(10));");

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

```
    }  
    e.printStackTrace();  
  }  
}
```

Run the preprocessing statement to insert data in batches.

```
public static void BatchInsertData(Connection conn) {  
    PreparedStatement pst = null;  
  
    try {  
        //Generate a prepared statement.  
        pst = conn.prepareStatement("INSERT INTO jdbc_test1 VALUES (?,?)");  
        for (int i = 0; i < 100; i++) {  
            //Add parameters.  
            pst.setInt(1, i);  
            pst.setString(2, "data " + i);  
            pst.addBatch();  
        }  
        //Run batch processing.  
        pst.executeBatch();  
        pst.close();  
    } catch (SQLException e) {  
        if (pst != null) {  
            try {  
                pst.close();  
            } catch (SQLException e1) {  
                e1.printStackTrace();  
            }  
        }  
        e.printStackTrace();  
    }  
}
```

Run the precompilation statement to update data.

```
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;  
}
```

Run a query statement and retry upon a failure. The number of retry times can be configured.

```
public static void ExecPreparedSQL(Connection conn) throws InterruptedException {
    int maxRetryTime = 50;
    int time = 0;
    String result = null;
    do {
        time++;
        try {
            System.out.println("time:" + time);
            boolean ret = QueryRedo(conn);
            if(ret == false){
                System.out.println("retry, time:" + time);
                Thread.sleep(10000);
                QueryRedo(conn);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    } while (null == result && time < maxRetryTime);
}

/**
 * Main process. Call static methods one by one.
 * @param args
 * @throws InterruptedException
 */
public static void main(String[] args) throws InterruptedException {
    //Establish a connection to the database.
    Connection conn = GetConnection("testuser", "test@123");

    //Create a table.
    CreateTable(conn);

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

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

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

Importing and Exporting Data Through Local Files

When the JAVA language is used for secondary development based on DWS, you can use the CopyManager interface to export data from the database to a local file or import a local file to the database by streaming. The file can be in CSV or TEXT format.

The sample program is as follows. Load the DWS JDBC driver before running it.

gsjdbc4.jar is used as an example.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.io.IOException;
```



```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.sql.SQLException;
import org.postgresql.copy.CopyManager;
import org.postgresql.core.BaseConnection;

public class Copy{

    public static void main(String[] args)
    {
        String urls = new String("jdbc:postgresql://10.180.155.74:8000/gaussdb"); //URL of the database
        String username = new String("jack"); //Username
        String password = new String("*****"); //Password
        String tablename = new String("migration_table"); //Define table information.
        String tablename1 = new String("migration_table_1"); //Define table information.
        String driver = "org.postgresql.Driver";
        Connection conn = null;

        try {
            Class.forName(driver);
            conn = DriverManager.getConnection(urls, username, password);
        } catch (ClassNotFoundException e) {
            e.printStackTrace(System.out);
        } catch (SQLException e) {
            e.printStackTrace(System.out);
        }
    }
}
```

Import and export data.

```
//Export the query result of migration_table to the local file d:/data.txt.
try {
    copyToFile(conn, "d:/data.txt", "(SELECT * FROM migration_table)");
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

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

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

public static void copyFromFile(Connection connection, String filePath, String tableName)
    throws SQLException, IOException {

    FileInputStream fileInputStream = null;

    try {
        CopyManager copyManager = new CopyManager((BaseConnection)connection);
        fileInputStream = new FileInputStream(filePath);
    }
}
```

```
        copyManager.copyIn("COPY " + tableName + " FROM STDIN", fileInputStream);
    } finally {
        if (fileInputStream != null) {
            try {
                fileInputStream.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

public static void copyToFile(Connection connection, String filePath, String tableOrQuery)
    throws SQLException, IOException {

    FileOutputStream fileOutputStream = null;

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

Migrating Data from MySQL to DWS

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

gsjdbc4.jar is used as an example.

```
import java.io.StringReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

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

public class Migration{

    public static void main(String[] args) {
        String url = new String("jdbc:postgresql://10.180.155.74:8000/gaussdb"); //URL of the database
        String user = new String("jack"); //DWS username
        String pass = new String("*****"); //DWS password
        String tablename = new String("migration_table"); //Define table information.
        String delimiter = new String("|"); //Define a delimiter.
        String encoding = new String("UTF8"); //Define a character set.
        String driver = "org.postgresql.Driver";
        StringBuffer buffer = new StringBuffer(); //Define the buffer to store formatted data.

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

            //Traverse the result set and obtain records row by row.
            //The values of columns in each record are separated by the specified delimiter and end with a
```

```
newline character to form strings.
    ///Add the strings to the buffer.
    while (rs.next()) {
        buffer.append(rs.getString(1) + delimiter
            + rs.getString(2) + delimiter
            + rs.getString(3) + delimiter
            + rs.getString(4)
            + "\n");
    }
    rs.close();

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

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

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

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

Return the query result from the source database.

```
private static ResultSet getDataSet() {
    ResultSet rs = null;
    try {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        Connection conn = DriverManager.getConnection("jdbc:mysql://10.119.179.227:3306/jack?
useSSL=false&allowPublicKeyRetrieval=true", "jack", "*****");
        Statement stmt = conn.createStatement();
        rs = stmt.executeQuery("select * from migration_table");
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return rs;
}
```

13.2.8 Processing RoaringBitmap Result Sets and Importing It to DWS

DWS 8.1.3 and later versions support the RoaringBitmap function. When using the Java language to perform secondary development based on DWS, you can use the CopyManager interface to import a small amount of RoaringBitmap data to DWS.

 NOTE

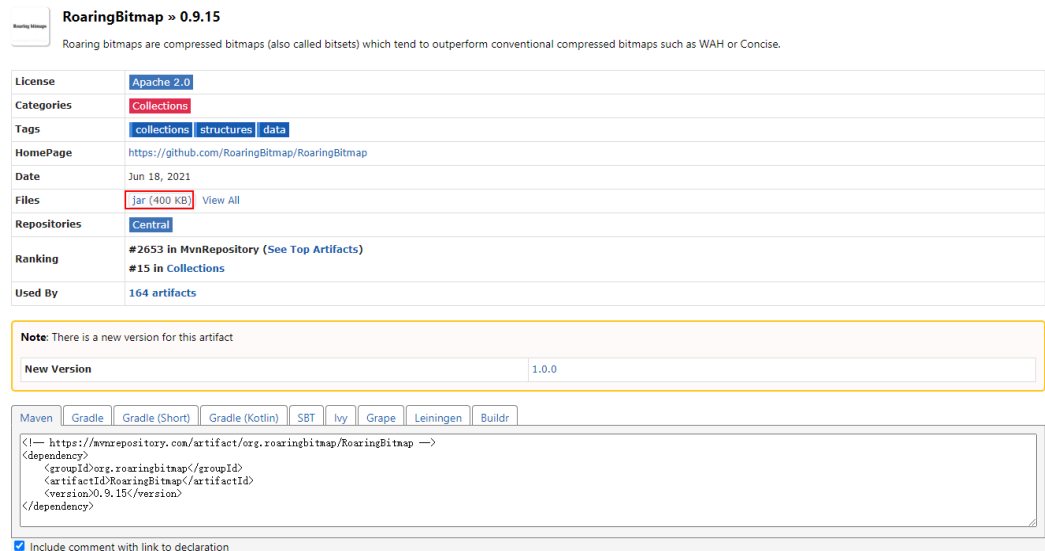
- Computing capability needs to be improved on the application side if a large amount of data needs to be imported to the database. Otherwise, the import performance will be affected.
- The 64-bit Roaringbitmap generated by Java cannot be imported to the database.

Processing RoaringBitmap Data

Step 1 Visit [Maven](#) to download the open-source RoaringBitmap JAR package. Version 0.9.15 is recommended.

The dependency items of the POM file are configured as follows:

```
<dependencies>
<dependency>
<groupId>org.roaringbitmap</groupId>
<artifactId>RoaringBitmap</artifactId>
<version>0.9.15</version>
</dependency>
</dependencies>
```



RoaringBitmap » 0.9.15
Roaring bitmaps are compressed bitmaps (also called bitsets) which tend to outperform conventional compressed bitmaps such as WAH or Concise.

License	Apache 2.0
Categories	Collections
Tags	collections structures data
HomePage	https://github.com/RoaringBitmap/RoaringBitmap
Date	Jun 18, 2021
Files	jar (400 KB) View All
Repositories	Central
Ranking	#2653 in MvnRepository (See Top Artifacts) #15 in Collections
Used By	164 artifacts

Note: There is a new version for this artifact
New Version 1.0.0

Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/org.roaringbitmap/RoaringBitmap -->
<dependency>
<groupId>org.roaringbitmap</groupId>
<artifactId>RoaringBitmap</artifactId>
<version>0.9.15</version>
</dependency>
```

☒ Include comment with link to declaration

Step 2 Invoke the JAR package to convert data to the RoaringBitmap type.

The general process is to declare a Roaring bitmap, call the add() method to convert data of the int type into the Roaringbitmap type, and then serialize the converted data. The sample code is as follows:

```
RoaringBitmap rr2 = new RoaringBitmap ();
for (int i = 1; i < 100000000; i++) {
    rr2.add(i);
}
ByteArrayOutputStream a = new ByteArrayOutputStream();
DataOutputStream b = new DataOutputStream(a);
rr2.serialize(b);
```

----End

Data Import

Invoke CopyManager to import data to the database. In this way, a small amount of RoaringBitmap data can be imported to the database without having to be stored locally.

```
//gsjdbc4.jar is used as an example.

package rb_demo;

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

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
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 rb_demo {

    private static String hexStr = "0123456789ABCDEF";

    public static String bytesToHex(byte[] bytes) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < bytes.length; i++) {
            String hex = Integer.toHexString(bytes[i] & 0xFF);
            if (hex.length() < 2) {
                sb.append(0);
            }
            sb.append(hex);
        }
        return sb.toString();
    }

    public static Connection GetConnection(String username, String passwd) {
        String driver = "org.postgresql.Driver";
        String sourceURL = "jdbc:postgresql://10.185.180.161: 8000/gaussdb"; //Database URL
        Connection conn = null;
        try {
            //Load the database driver.
            Class.forName(driver).newInstance();
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }

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

        return conn;
    }

    public static void main(String[] args) throws IOException {
```

```
RoaringBitmap rr2 = new RoaringBitmap();

for (int i = 1; i < 10000000; i++) {
    rr2.add(i);
}

ByteArrayOutputStream a = new ByteArrayOutputStream();

DataOutputStream b = new DataOutputStream(a);
rr2.serialize(b);

Connection conn = GetConnection("test", "password"); //Username and password
Statement pstmt = null;
try {
    conn.setAutoCommit(true);
    pstmt = conn.createStatement();

    pstmt.execute("drop table if exists t_rb");
    pstmt.execute("create table t_rb(c1 int, c2 roaringbitmap) distribute by hash (c1);");

    StringReader sr = null;
    CopyManager cm = null;
    cm = new CopyManager((BaseConnection) conn);

    String delimiter = "|";
    StringBuffer tuples = new StringBuffer();
    tuples.append("1" + delimiter + "\\x" + bytesToHex(a.toByteArray()));

    StringBuffer sb = new StringBuffer();
    sb.append(tuples.toString());

    sr = new StringReader(tuples.toString());
    String sql = "copy t_rb from STDIN with (delimiter '|', NOESCAPING)";

    long rows = cm.copyIn(sql, sr); //Execute the COPY command to save data to the database.

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

13.2.9 JDBC Interfaces

JDBC interface is a set of API methods for users. This section describes some common interfaces. For other interfaces, see information in JDK1.6 (software package) and JDBC 4.0.

java.sql.Connection

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

Table 13-7 java.sql.Connection methods

Method	Return Type	Support JDBC 4 or Not
close()	void	Yes
commit()	void	Yes
createStatement()	Statement	Yes
getAutoCommit()	boolean	Yes
getClientInfo()	Properties	Yes
getClientInfo(String name)	String	Yes
getTransactionIsolation()	int	Yes
isClosed()	boolean	Yes
isReadOnly()	boolean	Yes
prepareStatement(String sql)	PreparedStatement	Yes
rollback()	void	Yes
setAutoCommit(boolean autoCommit)	void	Yes
setClientInfo(Properties properties)	void	Yes
setClientInfo(String name,String value)	void	Yes

NOTICE

The interface uses the AutoCommit mode by default, but you can disable it by setting **setAutoCommit** to **false**. This will package all subsequent statements in explicit transactions. Note that you will not be able to execute statements that cannot be executed within transactions.

java.sql.CallableStatement

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

Table 13-8 java.sql.CallableStatement methods

Method	Return Type	Support JDBC 4 or Not
registerOutParameter(int parameterIndex, int type)	void	Yes

Method	Return Type	Support JDBC 4 or Not
wasNull()	boolean	Yes
getString(int parameterIndex)	String	Yes
getBoolean(int parameterIndex)	boolean	Yes
getByte(int parameterIndex)	byte	Yes
getShort(int parameterIndex)	short	Yes
getInt(int parameterIndex)	int	Yes
getLong(int parameterIndex)	long	Yes
getFloat(int parameterIndex)	float	Yes
getDouble(int parameterIndex)	double	Yes
getBigDecimal(int parameterIndex)	BigDecimal	Yes
getBytes(int parameterIndex)	byte[]	Yes
getDate(int parameterIndex)	Date	Yes
getTime(int parameterIndex)	Time	Yes
getTimestamp(int parameterIndex)	Timestamp	Yes
getObject(int parameterIndex)	Object	Yes

 NOTE

- Do not perform batch operations on statements containing **OUT** parameters.
- 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**.

java.sql.DatabaseMetaData

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

Table 13-9 java.sql.DatabaseMetaData methods

Method	Return Type	Support JDBC 4 or Not
getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)	ResultSet	Yes
getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)	ResultSet	Yes
getTableTypes()	ResultSet	Yes
getUserName()	String	Yes
isReadOnly()	boolean	Yes
nullsAreSortedHigh()	boolean	Yes
nullsAreSortedLow()	boolean	Yes
nullsAreSortedAtStart()	boolean	Yes
nullsAreSortedAtEnd()	boolean	Yes
getDatabaseProductName()	String	Yes
getDatabaseProductVersion()	String	Yes
getDriverName()	String	Yes
getDriverVersion()	String	Yes
getDriverMajorVersion()	int	Yes
getDriverMinorVersion()	int	Yes
usesLocalFiles()	boolean	Yes
usesLocalFilePerTable()	boolean	Yes
supportsMixedCaseIdentifiers()	boolean	Yes
storesUpperCaseIdentifiers()	boolean	Yes
storesLowerCaseIdentifiers()	boolean	Yes
supportsMixedCaseQuotedIdentifiers()	boolean	Yes
storesUpperCaseQuotedIdentifiers()	boolean	Yes
storesLowerCaseQuotedIdentifiers()	boolean	Yes

Method	Return Type	Support JDBC 4 or Not
storesMixedCaseQuotedIdentifiers()	boolean	Yes
supportsAlterTableWithAddColumn()	boolean	Yes
supportsAlterTableWithDropColumn()	boolean	Yes
supportsColumnAliasing()	boolean	Yes
nullPlusNonNullIsNull()	boolean	Yes
supportsConvert()	boolean	Yes
supportsConvert(int fromType, int toType)	boolean	Yes
supportsTableCorrelationNames()	boolean	Yes
supportsDifferentTableCorrelationNames()	boolean	Yes
supportsExpressionsInOrderBy()	boolean	Yes
supportsOrderByUnrelated()	boolean	Yes
supportsGroupBy()	boolean	Yes
supportsGroupByUnrelated()	boolean	Yes
supportsGroupByBeyondSelect()	boolean	Yes
supportsLikeEscapeClause()	boolean	Yes
supportsMultipleResultSets()	boolean	Yes
supportsMultipleTransactions()	boolean	Yes
supportsNonNullableColumns()	boolean	Yes
supportsMinimumSQLGrammar()	boolean	Yes
supportsCoreSQLGrammar()	boolean	Yes
supportsExtendedSQLGrammar()	boolean	Yes
supportsANSI92EntryLevelSQL()	boolean	Yes

Method	Return Type	Support JDBC 4 or Not
supportsANSI92IntermediateSQL()	boolean	Yes
supportsANSI92FullSQL()	boolean	Yes
supportsIntegrityEnhancementFacility()	boolean	Yes
supportsOuterJoins()	boolean	Yes
supportsFullOuterJoins()	boolean	Yes
supportsLimitedOuterJoins()	boolean	Yes
isCatalogAtStart()	boolean	Yes
supportsSchemasInDataManipulation()	boolean	Yes
supportsSavepoints()	boolean	Yes
supportsResultSetHoldability(int holdability)	boolean	Yes
getResultSetHoldability()	int	Yes
getDatabaseMajorVersion()	int	Yes
getDatabaseMinorVersion()	int	Yes
getJDBCMinorVersion()	int	Yes
getJDBCMajorVersion()	int	Yes

java.sql.Driver

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

Table 13-10 java.sql.Driver methods

Method	Return Type	Support JDBC 4 or Not
acceptsURL(String url)	boolean	Yes
connect(String url, Properties info)	Connection	Yes
jdbcCompliant()	boolean	Yes
getMajorVersion()	int	Yes
getMinorVersion()	int	Yes

java.sql.PreparedStatement

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

Table 13-11 java.sql.PreparedStatement methods

Method	Return Type	Support JDBC 4 or Not
clearParameters()	void	Yes
execute()	boolean	Yes
executeQuery()	ResultSet	Yes
executeUpdate()	int	Yes
getMetaData()	ResultSetMetaData	Yes
setBoolean(int parameterIndex, boolean x)	void	Yes
setBigDecimal(int parameterIndex, BigDecimal x)	void	Yes
setByte(int parameterIndex, byte x)	void	Yes
setBytes(int parameterIndex, byte[] x)	void	Yes
setDate(int parameterIndex, Date x)	void	Yes
setDouble(int parameterIndex, double x)	void	Yes
setFloat(int parameterIndex, float x)	void	Yes
setInt(int parameterIndex, int x)	void	Yes
setLong(int parameterIndex, long x)	void	Yes
setNString(int parameterIndex, String value)	void	Yes
setShort(int parameterIndex, short x)	void	Yes

Method	Return Type	Support JDBC 4 or Not
setString(int parameterIndex, String x)	void	Yes
addBatch()	void	Yes
executeBatch()	int[]	Yes
clearBatch()	void	Yes

 NOTE

- **addBatch()** and **execute()** can be executed only after **clearBatch()**.
- Calling the **executeBatch()** method does not clear the batch. Clear batch by explicitly calling **clearBatch()**.
- You do not need to use **set*()** to reuse the values of bounded variables in a batch after they have been added.
- The following methods are inherited from **java.sql.Statement**: **close**, **execute**, **executeQuery**, **executeUpdate**, **getConnection**, **getResultSet**, **getUpdateCount**, **isClosed**, **setMaxRows**, and **setFetchSize**.

java.sql.ResultSet

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

Table 13-12 java.sql.ResultSet methods

Method	Return Type	Support JDBC 4 or Not
findColumn(String columnLabel)	int	Yes
getBigDecimal(int columnIndex)	BigDecimal	Yes
getBigDecimal(String columnLabel)	BigDecimal	Yes
getBoolean(int columnIndex)	boolean	Yes
getBoolean(String columnLabel)	boolean	Yes
getByte(int columnIndex)	byte	Yes
getBytes(int columnIndex)	byte[]	Yes
getByte(String columnLabel)	byte	Yes

Method	Return Type	Support JDBC 4 or Not
getBytes(String columnLabel)	byte[]	Yes
getDate(int columnIndex)	Date	Yes
getDate(String columnLabel)	Date	Yes
getDouble(int columnIndex)	double	Yes
getDouble(String columnLabel)	double	Yes
getFloat(int columnIndex)	float	Yes
getFloat(String columnLabel)	float	Yes
getInt(int columnIndex)	int	Yes
getInt(String columnLabel)	int	Yes
getLong(int columnIndex)	long	Yes
getLong(String columnLabel)	long	Yes
getShort(int columnIndex)	short	Yes
getShort(String columnLabel)	short	Yes
getString(int columnIndex)	String	Yes
getString(String columnLabel)	String	Yes
getTime(int columnIndex)	Time	Yes
getTime(String columnLabel)	Time	Yes
getTimestamp(int columnIndex)	Timestamp	Yes
getTimestamp(String columnLabel)	Timestamp	Yes
isAfterLast()	boolean	Yes
isBeforeFirst()	boolean	Yes
isFirst()	boolean	Yes

Method	Return Type	Support JDBC 4 or Not
next()	boolean	Yes

 NOTE

- A statement cannot have multiple open result sets.
- The cursor used to traverse the result set cannot remain in the open state after being committed.

java.sql.ResultSetMetaData

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

Table 13-13 java.sql.ResultSetMetaData methods

Method	Return Type	Support JDBC 4 or Not
getColumnCount()	int	Yes
getColumnName(int column)	String	Yes
getColumnType(int column)	int	Yes
getColumnTypeName(int column)	String	Yes

java.sql.Statement

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

Table 13-14 java.sql.Statement methods

Method	Return Type	Support JDBC 4 or Not
close()	void	Yes
execute(String sql)	boolean	Yes
executeQuery(String sql)	ResultSet	Yes
executeUpdate(String sql)	int	Yes
getConnection()	Connection	Yes

Method	Return Type	Support JDBC 4 or Not
getResultSet()	ResultSet	Yes
getQueryTimeout()	int	Yes
getUpdateCount()	int	Yes
isClosed()	boolean	Yes
setQueryTimeout(int seconds)	void	Yes
setFetchSize(int rows)	void	Yes
cancel()	void	Yes

 NOTE

setFetchSize can reduce the memory occupied by the result set on the client. Result sets are packaged into cursors and segmented for processing, which will increase the communication traffic between the database and the client, affecting performance.

Database cursors are valid only within their transactions. Therefore, when setting **setFetchSize**, set **setAutoCommit** to **false** and commit transactions on the connection to flush service data to a database.

javax.sql.ConnectionPoolDataSource

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

Table 13-15 javax.sql.ConnectionPoolDataSource methods

Method	Return Type	Support JDBC 4 or Not
getLoginTimeout()	int	Yes
getLogWriter()	PrintWriter	Yes
getPooledConnection()	PooledConnection	Yes
getPooledConnection(String user,String password)	PooledConnection	Yes
setLoginTimeout(int seconds)	void	Yes
setLogWriter(PrintWriter out)	void	Yes

javax.sql.DataSource

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

Table 13-16 javax.sql.DataSource methods

Method	Return Type	Support JDBC 4 or Not
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

javax.sql.PooledConnection

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

Table 13-17 javax.sql.PooledConnection methods

Method	Return Type	Support JDBC 4 or Not
addConnectionEventListener (ConnectionEventListener listener)	void	Yes
close()	void	Yes
getConnection()	Connection	Yes
removeConnectionEventListener (ConnectionEventListener listener)	void	Yes
addStatementEventListener (StatementEventListener listener)	void	Yes
removeStatementEventListener (StatementEventListener listener)	void	Yes

javax.naming.Context

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

Table 13-18 javax.naming.Context methods

Method	Return Type	Support JDBC 4 or Not
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

javax.naming.spi.InitialContextFactory

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

Table 13-19 javax.naming.spi.InitialContextFactory methods

Method	Return Type	Support JDBC 4 or Not
getInitialContext(Hashtable<?,?> environment)	Context	Yes

CopyManager

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

Inheritance relationship of CopyManager

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

```
public class CopyManager
extends Object
```

Construction method

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

Common methods**Table 13-20** Common methods of CopyManager

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

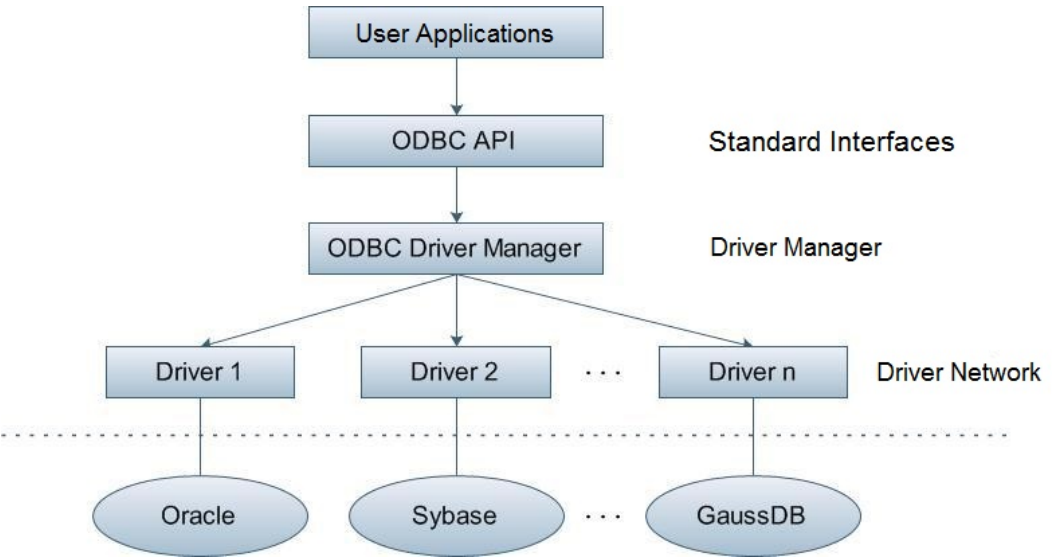
Returned Value	Method	Description	throws
long	copyOut(String sql, Writer to)	Sends the result set of COPY TO STDOUT from the database to the Writer class.	SQLException,IOException

13.3 ODBC-Based Development

Open Database Connectivity (ODBC) is a Microsoft API for accessing databases based on the X/OPEN CLI. The ODBC API alleviates applications from directly operating in databases, and enhances the database portability, extensibility, and maintainability.

Figure 13-2 shows the system structure of ODBC.

Figure 13-2 ODBC system structure



DWS supports ODBC 3.5 in the following environments.

Table 13-21 OSs Supported by ODBC

OS	Platform
SUSE Linux Enterprise Server 11 SP1/SP2/SP3/SP4 SUSE Linux Enterprise Server 12 and SP1/SP2/SP3/SP5	x86_64
Red Hat Enterprise Linux 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4/7.5	x86_64

OS	Platform
Red Hat Enterprise Linux 7.5	ARM64
CentOS 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4	x86_64
CentOS 7.6	ARM64
EulerOS 2.0 SP2/SP3	x86_64
EulerOS 2.0 SP8	ARM64
Neokylin 7.5/7.6	ARM64
Oracle Linux R7U4	x86_64
Windows 7	32-bit
Windows 7	64-bit
Windows Server 2008	32-bit
Windows Server 2008	64-bit

The operating systems listed above refer to the operating systems on which the ODBC program runs. They can be different from the operating systems where databases are deployed.

The ODBC Driver Manager running on UNIX or Linux can be unixODBC or iODBC. Select unixODBC-2.3.0 here as the component for connecting the database.

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

NOTE

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

13.3.1 ODBC Package and Its Dependent Libraries and Header Files

Download the ODBC software package from the console.

For details, see "Downloading the JDBC or ODBC Driver" in the *Data Warehouse Service (DWS) User Guide*.

ODBC Package for the Linux OS

Obtain the **dws_8.x.x_odbc_driver_for_xxx_xxx.zip** package from the software package. In the Linux OS, header files (including **sql.h** and **sql.h**) and library (**libodbc.so**) are required in application development. These header files and libraries can be obtained from the unixODBC-2.3.0 installation package.

ODBC Package for the Windows OS

Obtain the **dws_8.x.x_odbc_driver_for_windows.zip** package from the software package. In the Windows OS, the required header files and library files are system-resident.

13.3.2 Configuring a Data Source in the Linux OS

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

Procedure

Step 1 Obtain the source code package of unixODBC at:

<https://sourceforge.net/projects/unixodbc/files/unixODBC/2.3.0/unixODBC-2.3.0.tar.gz/download>

Step 2 Currently, unixODBC-2.2.1 is not supported. Assume you are to install unixODBC-2.3.0. Run the following commands. When installing, you can use **--prefix=[your_path]** to specify the installation directory. The data source file will be created in **[your_path]/etc**, and the library file will be generated in **[your_path]/lib**.

```
tar zxvf unixODBC-2.3.0.tar.gz
cd unixODBC-2.3.0
# Open the configure file. If it does not exist, open the configure.ac file. Find LIB_VERSION.
# Change the value of LIB_VERSION to 1:0:0 to compile a *.so.1 dynamic library with the same dependency on psqlodbcw.so.
vim configure

./configure --enable-gui=no --prefix=[your_path] #To perform the compilation on an Arm server, add the
configure parameter --build=aarch64-unknown-linux-gnu.
make
make install
```

Install unixODBC. If another version of unixODBC has been installed, it will be overwritten after installation.

Step 3 Replace the DWS client driver.

Decompress **dws_8.x.x_odbc_driver_for_XXX_XXX.zip** to obtain the **psqlodbcw.la** and **psqlodbcw.so** files in the **/dws_8.x.x_odbc_driver_for_XXX_XXX/odbc/lib** directory.

Step 4 Configure the data source.

1. Configure the ODBC driver file.

Add the following content to the **[your_path]/etc/odbcinst.ini** file:

```
[GaussMPP]
Driver64=[your_path]/lib/odbc/psqlodbcw.so
setup=[your_path]/lib/odbc/psqlodbcw.so
```

For descriptions of the parameters in the **odbcinst.ini** file, see [Table 13-22](#).

Table 13-22 odbcinst.ini configuration parameters

Parameter	Description	Examples
[DriverName]	Driver name, corresponding to Driver in DSN.	[DRIVER_N]
Driver64	Path of the dynamic driver library	Driver64=/xxx/odbc/lib/odbc/psqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/xxx/odbc/lib/odbc/psqlodbcw.so

2. Configure the data source file.

Add the following content to the **[your_path]/etc/odbc.ini** file:

```
[MPPODBC]
Driver=GaussMPP
Servername=10.10.0.13 (database server IP address)
Database=gaussdb (database name)
Username=dbadmin (database username)
Password= (database user password)
Port=8000 (database listening port)
Sslmode=allow
```

For descriptions of the parameters in the **odbc.ini** file, see [Table 13-23](#).

Table 13-23 odbc.ini configuration parameters

Parameter	Description	Examples
[DSN]	Data source name	[MPPODBC]
Driver	Driver name, corresponding to DriverName in odbcinst.ini	Driver=DRIVER_N
Servername	IP address of the server	Servername=10.145.130.26
Database	Name of the database to connect to	Database=gaussdb
Username	Name of the database user	Username=dbadmin

Parameter	Description	Examples
Password	Password of the database user	Password= NOTE After a user established a connection, the ODBC driver automatically clears their password stored in memory. However, if this parameter is configured, UnixODBC will cache data source files, which may cause the password to be stored in the memory for a long time. When you connect to an application, you are advised to send your password through an API instead of writing it in a data source configuration file. After the connection has been established, immediately clear the memory segment where your password is stored.
Port	Port ID of the server	Port=8000
Sslmode	Whether to enable the SSL	Sslmode=allow
UseServerSidePrepare	Whether to enable the extended query protocol for the database. The value can be 0 or 1 . The default value is 1 , indicating that the extended query protocol is enabled.	UseServerSidePrepare=1

Parameter	Description	Examples
UseBatchProtocol	Whether to enable the batch query protocol. If it is enabled, the DML performance can be improved. The value can be 0 or 1. The default value is 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 the support_batch_bind parameter is set to on, the batch query protocol is enabled.	UseBatchProtocol=1
ConnectionExtraInfo	Whether to display the driver deployment path and process owner in the connection_info parameter mentioned in connection_info	ConnectionExtraInfo=1 NOTE The default value is 1. If this parameter is set to 0, the ODBC driver reports the name and version of the current driver to the database. If this parameter is set to 1, the ODBC driver reports the name, deployment path, and process owner of the current driver to the database and records them in the connection_info parameter (see connection_info). You can query this parameter in PG_STAT_ACTIVITY and PGXC_STAT_ACTIVITY.

Parameter	Description	Examples
ForExtensionConnector	ETL tool performance optimization parameter. It can be used to optimize the memory and reduce the memory usage by the peer CN, to avoid system instability caused by excessive CN memory usage. The value can be 0 or 1. The default value is 0, indicating that the optimization item is disabled. Do not set this parameter for other services outside the database system. Otherwise, the service correctness may be affected.	ForExtensionConnector=1
KeepDisallowPremature	Specifies whether the cursor in the SQL statement has the with hold attribute when the following conditions are met: UseDeclareFetch is set to 1, and the application invokes SQLNumResultCols, SQLDescribeCol, or SQLColAttribute after invoking SQLPrepare to obtain the column information of the result set. The value can be 0 or 1. 0 indicates that the with hold attribute is supported, and 1 indicates that the with hold attribute is not supported. The default value is 0.	KeepDisallowPremature=1 NOTE When UseServerSidePrepare is set to 1, the KeepDisallowPremature parameter does not take effect. To use this parameter, set UseServerSidePrepare to 0. For example, set UseDeclareFetch to 1. KeepDisallowPremature=1 UseServerSidePrepare=0

The valid values of **sslmode** are as follows.

Table 13-24 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	The SSL secure connection is not used.
allow	Probably	The SSL secure encrypted connection is used if required by the database server, but does not check the authenticity of the server.
prefer	Probably	The SSL secure encrypted connection is used as a preferred mode if supported by the database, but does not check the authenticity of the server.
require	Yes	The SSL secure connection must be used, but it only encrypts data and does not check the authenticity of the server.
verify-ca	Yes	The SSL secure connection must be used, and it checks whether the database has certificates issued by a trusted CA.
verify-full	Yes	The SSL secure connection must be used. In addition to the check scope specified by verify-ca , it checks whether the name of the host where the database resides is the same as that on the certificate. This mode is not supported.

Step 5 Enable the SSL mode.

To use SSL certificates for connection, decompress the certificate package contained in the DWS installation package, and run **source sslcert_env.sh** in a shell environment to deploy certificates in the default location of the current session.

Or manually declare the following environment variables and ensure that the permission for the client.key* series files is set to 600.

```
export PGSSLCERT= "/YOUR/PATH/OF/client.crt" # Change the path to the absolute path of client.crt.  
export PGSSLKEY= "/YOUR/PATH/OF/client.key" # Change the path to the absolute path of client.key.
```

In addition, change the value of **Sslmode** in the data source to **verify-ca**.

Step 6 Add the IP address segment of the host where the client is located to the security group rules of DWS to ensure that the host can communicate with DWS.**Step 7** Configure environment variables.

```
vim ~/.bashrc
```

Add the following content to the end of the configuration file:

```
export LD_LIBRARY_PATH=[your_path]/lib:$LD_LIBRARY_PATH  
export ODBCYSINI=[your_path]/etc  
export ODBCINI=[your_path]/etc/odbc.ini
```

 NOTE

It is not recommended to add **LD_LIBRARY_PATH** in the Kylin OS, as it may cause conflicts with the **libssl.so** dynamic library. In the latest version of cluster 9.1.0, the **rpath** mechanism has been added, so the dependency can be located without **LD_LIBRARY_PATH**.

Step 8 Run the following commands to validate the settings:

```
source ~/.bashrc
```

----End

Testing Data Source Configuration

Run the **isql-v GaussODBC** command (*GaussODBC* is the data source name).

- If the following information is displayed, the configuration is correct and the connection succeeds.

```
+-----+
| Connected!                |
|                             |
| sql-statement             |
| help [tablename]          |
| quit                      |
|                             |
+-----+
SQL>
```

- If error information is displayed, the configuration is incorrect. Check the configuration.

Troubleshooting

- [UnixODBC][Driver Manager]Can't open lib 'xxx/xxx/psqlodbcw.so' : file not found.

Possible causes:

- The path configured in the **odbcinst.ini** file is incorrect.

Run **ls** to check the path in the error information, ensuring that the **psqlodbcw.so** file exists and you have execution permissions on it.

- The dependent library of **psqlodbcw.so** does not exist or is not in system environment variables.

Run **ldd** to check the path in the error information. If **libodbc.so.1** or other UnixODBC libraries are lacking, configure UnixODBC again following the procedure provided in this section, and add the **lib** directory under its installation directory to **LD_LIBRARY_PATH**. If other libraries are lacking, add the **lib** directory under the ODBC driver package to **LD_LIBRARY_PATH**. Alternatively, you can place the dependency library of **psqlodbcw.so** in the path corresponding to **rpath** of **psqlodbcw.so**. To view **rpath**, you can use the **readelf -d** command.

- [UnixODBC]connect to server failed: no such file or directory

Possible causes:

- An incorrect or unreachable database IP address or port was configured.
Check the **Servername** and **Port** configuration items in data sources.
- Server monitoring is improper.

If **Servername** and **Port** are correctly configured, ensure the proper network adapter and port are monitored based on database server configurations in the procedure in this section.

- Firewall and network gatekeeper settings are improper.

Check firewall settings, ensuring that the database communication port is trusted.

Check to ensure network gatekeeper settings are proper (if any).

- [unixODBC]The password-stored method is not supported.

Possible causes:

The **sslmode** configuration item is not configured in the data sources.

Solution:

Set it to **allow** or a higher level. For more details, see [Table 13-24](#).

- Server common name "xxxx" does not match host name "xxxxx"

Possible causes:

When **verify-full** is used for SSL encryption, the driver checks whether the host name in certificates is the same as the actual one.

Solution:

To solve this problem, use **verify-ca** to stop checking host names, or generate a set of CA certificates containing the actual host names.

- Driver's SQLAllocHandle on SQL_HANDLE_DBC failed

Possible causes:

The executable file (such as the **isql** tool of unixODBC) and the database driver (**psqlodbcw.so**) depend on different library versions of ODBC, such as **libodbc.so.1** and **libodbc.so.2**. You can verify this problem by using the following method:

```
ldd `which isql` | grep odbc  
ldd psqlodbcw.so | grep odbc
```

If the suffix digits of the outputs **libodbc.so** are different or indicate different physical disk files, this problem exists. Both **isql** and **psqlodbcw.so** load **libodbc.so**. If different physical files are loaded, different ODBC libraries with the same function list conflict with each other in a visible domain. As a result, the database driver cannot be loaded.

Solution:

Uninstall the unnecessary unixODBC, such as **libodbc.so.2**, and create a soft link with the same name and the **.so.2** suffix for the remaining **libodbc.so.1** library.

- FATAL: Forbid remote connection with trust method!

For security purposes, the CN forbids access from other nodes in the cluster without authentication.

To access the CN from inside the cluster, deploy the ODBC program on the machine where the CN is located and use 127.0.0.1 as the server address. It is recommended that the service system be deployed outside the cluster. If it is deployed inside, the database performance may be affected.

- [unixODBC][Driver Manager]Invalid attribute value

This problem occurs when you use SQL on other GaussDB. The possible cause is that the unixODBC version is not the recommended one. You are advised to run the **odbcinst --version** command to check the unixODBC version.

- authentication method 10 not supported.

If this error occurs on an open source client, the cause may be:

The database stores only the SHA-256 hash of the password, but the open source client supports only MD5 hashes.

NOTE

- The database stores the hashes of user passwords instead of actual passwords.
- In versions earlier than V100R002C80SPC300, the database stores only SHA-256 hashes and no MD5 hashes. Therefore, MD5 cannot be used for user password authentication.
- In V100R002C80SPC300 and later, if a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
- An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. If your database is upgraded from a version earlier than V100R002C80SPC300, passwords in the old version will only have SHA-256 hashes and not support MD5 authentication.

To solve this problem, you can update the user password. Alternatively, create a user, assign the same permissions to the user, and use the new user to connect to the database.

- unsupported frontend protocol 3.51: server supports 1.0 to 3.0

The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.

13.3.3 Configuring a Data Source in the Windows OS

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

Procedure

Step 1 Replace the DWS client driver.

Decompress **GaussDB-9.1.0-Windows-Odbc.tar.gz**. Double-click install **psqlodbc.msi** (for 32-bit OS) or **psqlodbc_x64.msi** (for 64-bit OS).

Step 2 Open Driver Manager.

Use the Driver Manager suitable for your OS to configure the data source. (Assume the Windows system drive is drive C.)

- If you develop 32-bit programs in the 64-bit Windows OS, open the 32-bit Driver Manager at **C:\Windows\SysWOW64\odbcad32.exe** after you install the 32-bit driver.

Do not open Driver Manager by choosing **Control Panel**, clicking **Administrative Tools**, and clicking **Data Sources (ODBC)**.

NOTE

WoW64 is the acronym for "Windows 32-bit on Windows 64-bit". **C:\Windows\SysWOW64** stores the 32-bit environment on a 64-bit system.

- If you develop 64-bit programs in the 64-bit Windows OS, open the 64-bit Driver Manager at **C:\Windows\System32\odbcad32.exe** after you install the 64-bit driver.

Do not open **Driver Manager** by choosing **Control Panel**, clicking **Administrative Tools**, and clicking **Data Sources (ODBC)**.

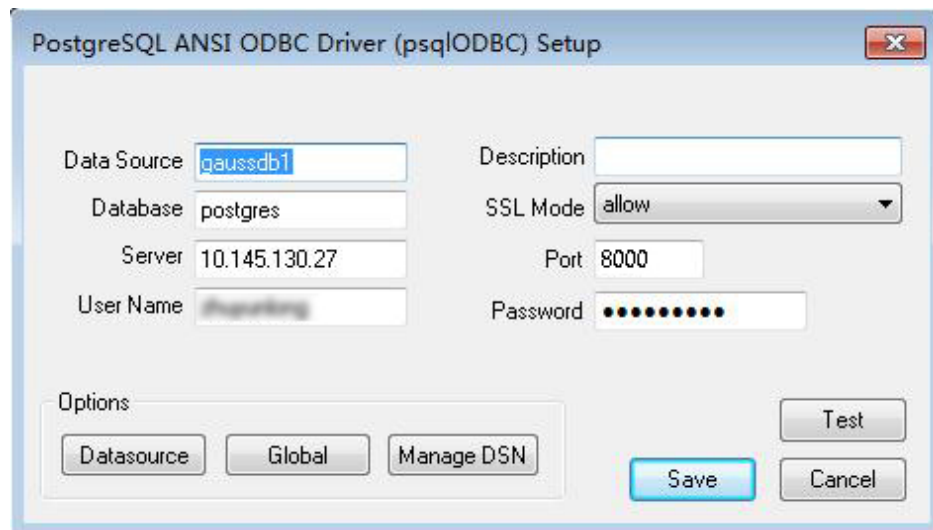
 NOTE

C:\Windows\System32 stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- In a 32-bit Windows OS, open **C:\Windows\System32\odbcad32.exe**.
In the Windows OS, click **Computer**, and choose **Control Panel**. Click **Administrative Tools** and click **Data Sources (ODBC)**.

Step 3 Configure the data source.

On the **User DSN** tab, click **Add**, and choose **PostgreSQL Unicode** for setup. (An identifier will be displayed for the 64-bit OS.)



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.

To use SSL certificates for connection, decompress the certificate package contained in the DWS installation package, and double-click the **sslcert_env.bat** file to deploy certificates in the default location.

NOTICE

The **sslcert_env.bat** file ensures the purity of the certificate environment. When the **%APPDATA%\postgresql** directory exists, a message will be prompted asking you whether you want to remove related directories. If you want to remove related directories, back up files in the directory.

Alternatively, you can copy the **client.crt**, **client.key**, **client.key.cipher**, and **client.key.rand** files in the certificate file folder to the manually created **%APPDATA%\postgresql** directory. Change **client** in the file names to **postgres**, for example, change **client.key** to **postgres.key**. Copy the **cacert.pem** file to the **%APPDATA%\postgresql** directory and change its name to **root.crt**.

Change the value of **SSL Mode** in step 2 to **verify-ca**.

Table 13-25 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	The SSL secure connection is not used.
allow	Probably	The SSL secure encrypted connection is used if required by the database server, but does not check the authenticity of the server.
prefer	Probably	The SSL secure encrypted connection is used as a preferred mode if supported by the database, but does not check the authenticity of the server.
require	Yes	The SSL secure connection must be used, but it only encrypts data and does not check the authenticity of the server.
verify-ca	Yes	The SSL secure connection must be used, and it checks whether the database has certificates issued by a trusted CA.
verify-full	Yes	The SSL secure connection must be used. In addition to the check scope specified by verify-ca , it checks whether the name of the host where the database resides is the same as that on the certificate. NOTE This mode cannot be used.

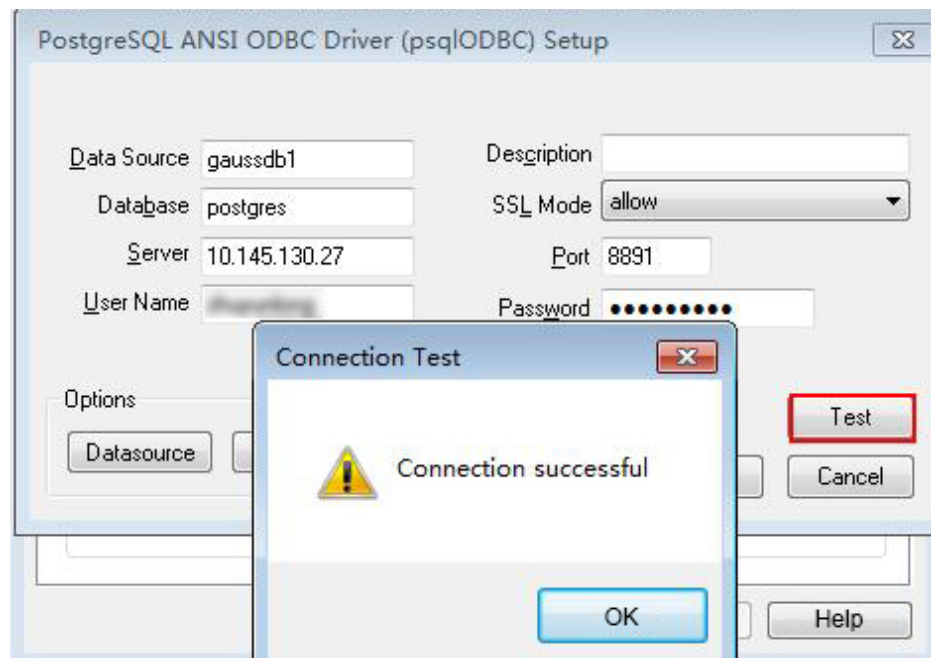
Step 5 Add the IP address segment of the host where the client is located to the security group rules of DWS to ensure that the host can communicate with DWS.

----End

Testing Data Source Configuration

Click **Test**.

- If the following information is displayed, the configuration is correct and the connection succeeds.



- If error information is displayed, the configuration is incorrect. Check the configuration.

Troubleshooting

- Server common name "xxxx" does not match host name "xxxxx"
This problem occurs because 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 CA certificates containing the actual host names.
- connect to server failed: no such file or directory
Possible causes:
 - An incorrect or unreachable database IP address or port was configured.
Check the **Servername** and **Port** configuration items in data sources.
 - Server monitoring is improper.
If **Servername** and **Port** are correctly configured, ensure the proper network adapter and port are monitored based on database server configurations in the procedure in this section.
 - Firewall and network gatekeeper settings are improper.
Check firewall settings, ensuring that the database communication port is trusted.
Check to ensure network gatekeeper settings are proper (if any).
- In the specified DSN, the system structures of the drive do not match those of the application.

Possible cause: The bit versions of the drive and program are different.

C:\Windows\SysWOW64\odbcad32.exe is a 32-bit ODBC Drive Manager.

C:\Windows\System32\odbcad32.exe is a 64-bit ODBC Drive Manager.

- The password-stored method is not supported.

Possible causes:

sslmode is not configured for the data source. Set this configuration item to **allow** or a higher level to enable SSL connections. For details about **sslmode**, see [Table 13-25](#).

- authentication method 10 not supported.

If this error occurs on an open source client, the cause may be:

The database stores only the SHA-256 hash of the password, but the open source client supports only MD5 hashes.

NOTE

- The database stores the hashes of user passwords instead of actual passwords.
- In versions earlier than V100R002C80SPC300, the database stores only SHA-256 hashes and no MD5 hashes. Therefore, MD5 cannot be used for user password authentication.
- In V100R002C80SPC300 and later, if a password is updated or a user is created, both types of hashes will be stored, compatible with open-source authentication protocols.
- An MD5 hash can only be generated using the original password, but the password cannot be obtained by reversing its SHA-256 hash. If your database is upgraded from a version earlier than V100R002C80SPC300, passwords in the old version will only have SHA-256 hashes and not support MD5 authentication.

To solve this problem, perform the following operations:

- a. Check and change the **password_encryption_type** to **1** if needed with the help of technical support.
 - b. Create a new database user for connection or reset the password of the existing database user.
 - If you are an administrator, reset the password by referring to "Resetting the Password of the DWS Database Administrator" in the *Data Warehouse Service (DWS) User Guide*.
 - If you are a common user, use another client tool (such as Data Studio) to connect to the database and run the **ALTER USER** statement to change your password.
 - c. Connect to the database.
- unsupported frontend protocol 3.51: server supports 1.0 to 3.0
The database version is too early or the database is an open-source database. Use the driver of the required version to connect to the database.
 - FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

In some cases, the error is: GSSAPI authentication not supported.

In **pg_hba.conf** of the target CN, the authentication mode is set to **gss** for authenticating the IP address of the current client. However, this authentication algorithm cannot authenticate clients. Change the authentication algorithm to **sha256** and try again.

Note that cross-node connection to the database in the cluster is not supported. If the error is caused by cross-node connection to the CN in the cluster, connect the service program to the database from a node outside the cluster and try again.

13.3.4 ODBC Development Example

Code for Common Functions

The following example shows how to obtain data from DWS through ODBC.

```
// DBtest.c (compile with: libodbc.so)
#include <stdlib.h>
#include <stdio.h>
#include <sqlext.h>
#ifdef WIN32
#include <windows.h>
#endif
SQLHENV    V_OD_Env;      // Handle ODBC environment
SQLHSTMT    V_OD_hstmt;   // Handle statement
SQLHDBC     V_OD_hdbc;    // Handle connection
char        typename[100];
SQLINTEGER  value = 100;
SQLINTEGER  V_OD_erg,V_OD_buffer,V_OD_err,V_OD_id;
SQLLEN      V_StrLen_or_IndPtr;
int main(int argc,char *argv[])
{
    // 1. Apply for an environment handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&V_OD_Env);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error AllocHandle\n");
        exit(0);
    }
    // 2. Set environment attributes (version information)
    SQLSetEnvAttr(V_OD_Env, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);
    // 3. Apply for a connection handle.
    V_OD_erg = SQLAllocHandle(SQL_HANDLE_DBC, V_OD_Env, &V_OD_hdbc);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    // 4. Set connection attributes.
    SQLSetConnectAttr(V_OD_hdbc, SQL_ATTR_AUTOCOMMIT, SQL_AUTOCOMMIT_ON, 0);
    // 5. Connect to the data source. userName and password indicate the username and password for
    // connecting to the database. Set them as needed.
    // If the username and password have been set in the odbc.ini file, you do not need to set userName or
    // password here, retaining "" for them. However, you are not advised to do so because the username and
    // password will be disclosed if the permission for odbc.ini is abused.
    V_OD_erg = SQLConnect(V_OD_hdbc, (SQLCHAR*) "gaussdb", SQL_NTS,
        (SQLCHAR*) "userName", SQL_NTS, (SQLCHAR*) "password", SQL_NTS);
    if ((V_OD_erg != SQL_SUCCESS) && (V_OD_erg != SQL_SUCCESS_WITH_INFO))
    {
        printf("Error SQLConnect %d\n",V_OD_erg);
        SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
        exit(0);
    }
    printf("Connected !\n");
    // 6. Set statement attributes
    SQLSetStmtAttr(V_OD_hstmt,SQL_ATTR_QUERY_TIMEOUT,(SQLPOINTER *)3,0);
    // 7. Apply for a statement handle
    SQLAllocHandle(SQL_HANDLE_STMT, V_OD_hdbc, &V_OD_hstmt);
    // 8. Executes an SQL statement directly
    SQLExecDirect(V_OD_hstmt,"drop table IF EXISTS customer_t1",SQL_NTS);
    SQLExecDirect(V_OD_hstmt,"CREATE TABLE customer_t1(c_customer_sk INTEGER, c_customer_name
    VARCHAR(32));",SQL_NTS);
```

```
SQLExecDirect(V_OD_hstmt,"insert into customer_t1 values(25,'li')",SQL_NTS);
// 9. Prepare for execution
SQLPrepare(V_OD_hstmt,"insert into customer_t1 values(?)",SQL_NTS);
// 10. Bind parameters
SQLBindParameter(V_OD_hstmt,1,SQL_PARAM_INPUT,SQL_C_SLONG,SQL_INTEGER,0,0,
    &value,0,NULL);
// 11. Execute the ready statement
SQLExecute(V_OD_hstmt);
SQLExecDirect(V_OD_hstmt,"select c_customer_sk from customer_t1",SQL_NTS);
// 12. Obtain the attributes of a certain column in the result set

SQLColAttribute(V_OD_hstmt,1,SQL_DESC_TYPE_NAME,typename,sizeof(typename),NULL,NULL);

printf("SQLColAttribute %s\n",typename);
// 13. Bind the result set
SQLBindCol(V_OD_hstmt,1,SQL_C_SLONG, (SQLPOINTER)&V_OD_buffer,150,
    (SQLLEN *)&V_StrLen_or_IndPtr);
// 14. Collect data using SQLFetch
V_OD_erg=SQLFetch(V_OD_hstmt);
// 15. Obtain and return data using SQLGetData
while(V_OD_erg != SQL_NO_DATA)
{
    SQLGetData(V_OD_hstmt,1,SQL_C_SLONG,(SQLPOINTER)&V_OD_id,0,NULL);
    printf("SQLGetData ----ID = %d\n",V_OD_id);
    V_OD_erg=SQLFetch(V_OD_hstmt);
};
printf("Done !\n");
// 16. Disconnect from the data source and release handles
SQLFreeHandle(SQL_HANDLE_STMT,V_OD_hstmt);
SQLDisconnect(V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_DBC,V_OD_hdbc);
SQLFreeHandle(SQL_HANDLE_ENV, V_OD_Env);
return(0);
}
```

Code for Batch Processing

- Enable **UseBatchProtocol** in the data source and set **support_batch_bind** to **on**.
- Use **CHECK_ERROR** to check and print error information.
- This example is used to interactively obtain the DSN, data volume to be processed, and volume of ignored data from users, and insert required data into the **test_odbc_batch_insert** table.

```
#include <stdio.h>
#include <stdlib.h>
#include <sql.h>
#include <sqlext.h>
#include <string.h>

#include "util.c"

void Exec(SQLHDBC hdbc, SQLCHAR* sql)
{
    SQLRETURN retcode;           // Return status
    SQLHSTMT hstmt = SQL_NULL_HSTMT; // Statement handle
    SQLCHAR loginfo[2048];

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
        hstmt, SQL_HANDLE_STMT);

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

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

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

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

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

    // Interactively obtain data source names.
    getStr("Please input your DSN", (char*)dsn, sizeof(dsn), 'N');
    // Interactively obtain the amount of data to be batch processed.
    getInt("batchCount", &batchCount, 'N', 1);
    do
    {
        // Interactively obtain the amount of batch processing data that is not inserted into the database.
        getInt("ignoreCount", &ignoreCount, 'N', 1);
        if (ignoreCount > batchCount)
        {
            printf("ignoreCount(%d) should be less than batchCount(%d)\n", ignoreCount, batchCount);
        }
    }while(ignoreCount > batchCount);

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

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

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

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

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

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

    // init table info.
```

```
Exec(hdbc, "drop table if exists test_odbc_batch_insert");
Exec(hdbc, "create table test_odbc_batch_insert(id int primary key, col varchar2(50))");

// The following code constructs the data to be inserted based on the data volume entered by users:
{
    SQLRETURN retcode;
    SQLHSTMT hstmtinesrt = SQL_NULL_HSTMT;
    int i;
    SQLCHAR *sql = NULL;
    SQLINTEGER *ids = NULL;
    SQLCHAR *cols = NULL;
    SQLLEN *bufLenIds = NULL;
    SQLLEN *bufLenCols = NULL;
    SQLUSMALLINT *operptr = NULL;
    SQLUSMALLINT *statusptr = NULL;
    SQLULEN process = 0;

    // Data is constructed by column. Each column is stored continuously.
    ids = (SQLINTEGER*)malloc(sizeof(ids[0]) * batchCount);
    cols = (SQLCHAR*)malloc(sizeof(cols[0]) * batchCount * 50);
    // Data size in each row for a column
    bufLenIds = (SQLLEN*)malloc(sizeof(bufLenIds[0]) * batchCount);
    bufLenCols = (SQLLEN*)malloc(sizeof(bufLenCols[0]) * batchCount);
    // Whether this row needs to be processed. The value is SQL_PARAM_IGNORE or SQL_PARAM_PROCEED.
    operptr = (SQLUSMALLINT*)malloc(sizeof(operptr[0]) * batchCount);
    memset(operptr, 0, sizeof(operptr[0]) * batchCount);
    // Processing result of the row
    // Note: In the database, a statement belongs to one transaction. Therefore, data is processed as a unit.
    // That is, either all data is inserted successfully or all data fails to be inserted.
    statusptr = (SQLUSMALLINT*)malloc(sizeof(statusptr[0]) * batchCount);
    memset(statusptr, 88, sizeof(statusptr[0]) * batchCount);

    if (NULL == ids || NULL == cols || NULL == bufLenCols || NULL == bufLenIds)
    {
        fprintf(stderr, "FAILED:\tmalloc data memory failed\n");
        goto exit;
    }

    for (int i = 0; i < batchCount; i++)
    {
        ids[i] = i;
        sprintf(cols + 50 * i, "column test value %d", i);
        bufLenIds[i] = sizeof(ids[i]);
        bufLenCols[i] = strlen(cols + 50 * i);
        operptr[i] = (i < ignoreCount) ? SQL_PARAM_IGNORE : SQL_PARAM_PROCEED;
    }

    // Allocate Statement Handle
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmtinesrt);
    CHECK_ERROR(retcode, "SQLAllocHandle(SQL_HANDLE_STMT)",
        hstmtinesrt, SQL_HANDLE_STMT);

    // Prepare Statement
    sql = (SQLCHAR*)"insert into test_odbc_batch_insert values(?, ?)";
    retcode = SQLPrepare(hstmtinesrt, (SQLCHAR*) sql, SQL_NTS);
    sprintf((char*)loginfo, "SQLPrepare log: %s", (char*)sql);
    CHECK_ERROR(retcode, loginfo, hstmtinesrt, SQL_HANDLE_STMT);

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

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

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

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

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

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

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

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

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

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

        for (int i = 0; i < batchCount; i++)
        {
            if (i < ignoreCount)
            {
                if (statusptr[i] != SQL_PARAM_UNUSED)
                {
                    sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_UNUSED", i, statusptr[i]);
                    CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
                }
            }
            else if (statusptr[i] != SQL_PARAM_SUCCESS)
            {
                sprintf(loginfo, "statusptr[%d](%d) != SQL_PARAM_SUCCESS", i, statusptr[i]);
                CHECK_ERROR(SQL_ERROR, loginfo, NULL, SQL_HANDLE_STMT);
            }
        }

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

```
exit:
    printf ("\nComplete.\n");

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

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

    return 0;
}
```

13.3.5 ODBC APIs

ODBC APIs are provided for users. This section covers common APIs. For more details on other APIs, see the ODBC Programmer's Reference on [the MSDN website](#).

SQLAllocEnv

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

SQLAllocConnect

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

SQLAllocHandle

Function

SQLAllocHandle allocates environment, connection, or statement handles. It replaces the ODBC 2.x functions **SQLAllocEnv**, **SQLAllocConnect**, and **SQLAllocStmt**.

Prototype

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

Parameters

Table 13-26 SQLAllocHandle parameters

Parameter	Description
HandleType	Handle type allocated by SQLAllocHandle . The value must be one of the following: • SQL_HANDLE_ENV (environment handle) • SQL_HANDLE_DBC (connection handle) • SQL_HANDLE_STMT (statement handle) • SQL_HANDLE_DESC (description handle) The handle application sequence is: SQL_HANDLE_ENV > SQL_HANDLE_DBC > SQL_HANDLE_STMT . The handle applied later depends on the handle applied prior to it.
InputHandle	Type of the new handle to be allocated. • If HandleType is SQL_HANDLE_ENV, the value is SQL_NULL_HANDLE. • If HandleType is SQL_HANDLE_DBC, this must be an environment handle. • If HandleType is SQL_HANDLE_STMT or SQL_HANDLE_DESC, it must be a connection handle.
OutputHandlePtr	Output parameter: Pointer to a buffer in which the handle returned for the newly allocated data structure is stored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLAllocHandle** returns **SQL_ERROR** when it is used to allocate a non-environment handle, it sets **OutputHandlePtr** to **SQL_NULL_HDBC**, **SQL_NULL_HSTMT**, or **SQL_NULL_HDESC**. The application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **InputHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLAllocStmt

In ODBC 3.x, **SQLAllocStmt** (a function in ODBC 2.x) was deprecated and replaced by **SQLAllocHandle**. For details, see [SQLAllocHandle](#).

SQLBindCol

Function

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

Prototype

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

Parameters

Table 13-27 SQLBindCol parameters

Parameter	Description
StatementHandle	Statement handle.
ColumnNumber	Number of the column to be bound. Column numbering begins at 0 and increases in ascending order. Column 0 functions as the bookmark. If no bookmark column is set, column numbering begins at 1 instead.
TargetType	The C data type in the buffer.
TargetValuePtr	Output parameter: pointer to the buffer bound with the column. The SQLFetch function returns data in the buffer. If TargetValuePtr is null, StrLen_or_IndPtr is a valid value.
BufferLength	Length of the buffer to which TargetValuePtr points, in bytes.
StrLen_or_IndPtr	Output parameter: pointer to the length or indicator of the buffer. If StrLen_or_IndPtr is null, no length or indicator is used.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Note

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLBindParameter

Function

SQLBindParameter binds a parameter flag in an SQL statement to a buffer.

Prototype

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

Parameters

Table 13-28 SQLBindParameter

Keyword	Description
StatementHandle	Statement handle.
ParameterNumber	Parameter marker number, starting at 1 and increasing in an ascending order.
InputOutputType	Input and output parameter types.
ValueTType	C data type of the parameter.
ParameterType	SQL data type of the parameter.
ColumnSize	Column size or the expression of the corresponding parameter marker.
DecimalDigits	Decimal number of the column or the expression of the corresponding parameter marker.
ParameterValuePtr	Pointer to the buffer for storing parameter data.

Keyword	Description
BufferLength	Length of the buffer to which the ParameterValuePtr points, in bytes.
StrLen_or_IndPtr	Pointer to the length or indicator of the buffer. If StrLen_or_IndPtr is null, no length or indicator is used.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLBindCol** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLColAttribute

Function

SQLColAttribute returns the descriptor information about a column in the result set.

Prototype

```
SQLRETURN SQLColAttribute(SQLHSTMT StatementHandle,  
                           SQLUSMALLINT ColumnNumber,  
                           SQLUSMALLINT FieldIdentifier,  
                           SQLPOINTER CharacterAttributePtr,  
                           SQLSMALLINT BufferLength,  
                           SQLSMALLINT *StringLengthPtr,  
                           SQLPOINTER NumericAttributePtr);
```

Parameters

Table 13-29 SQLColAttribute parameters

Parameter	Description
StatementHandle	Statement handle.

Parameter	Description
ColumnNumber	Column number of the field to be queried, starting at 1 and increasing in an ascending order.
FieldIdentifier	Field identifier of ColumnNumber in IRD.
CharacterAttributePtr	Output parameter: pointer to the buffer that returns FieldIdentifier field value.
BufferLength	• FieldIdentifier indicates the buffer length when it refers to an ODBC-defined field and CharacterAttributePtr points to a string or binary buffer. • Ignore this parameter if FieldIdentifier is an ODBC-defined field and CharacterAttributePtr points to an integer.
StringLengthPtr	Output parameter: pointer to a buffer in which the total number of valid bytes (for string data) is stored in * CharacterAttributePtr . Ignore the value of BufferLength if the data is not a string.
NumericAttributePtr	Output parameter: pointer to an integer buffer in which the value of the FieldIdentifier field in the ColumnNumber row of the IRD is returned.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLColAttribute** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLConnect

Function

SQLConnect establishes a connection between a driver and a data source. Using the connection handle, you can obtain crucial information like the program's

status, transaction processing status, and error messages after establishing a connection to the data source.

Prototype

```
SQLRETURN SQLConnect(SQLHDBC ConnectionHandle,  
                      SQLCHAR *ServerName,  
                      SQLSMALLINT NameLength1,  
                      SQLCHAR *UserName,  
                      SQLSMALLINT NameLength2,  
                      SQLCHAR *Authentication,  
                      SQLSMALLINT NameLength3);
```

Parameters

Table 13-30 SQLConnect parameters

Parameter	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle .
ServerName	Name of the data source to connect to.
NameLength1	Length of ServerName .
UserName	Database username in the data source.
NameLength2	Length of UserName .
Authentication	Password of the database user in the data source.
NameLength3	Length of Authentication .

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLConnect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLDisconnect

Function

SQLDisconnect closes the connection associated with the database connection handle.

Prototype

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

Parameters

Table 13-31 SQLDisconnect parameters

Parameter	Description
ConnectionHandle	Connection handle, obtained from SQLAllocHandle .

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLDisconnect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLExecDirect

Function

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

Prototype

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

Parameters

Table 13-32 SQLExecDirect parameters

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

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_NEED_DATA** indicates that there are not enough parameters provided to execute the SQL statement.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.

Precautions

If **SQLExecDirect** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLExecute

Function

When a statement includes a parameter marker, the **SQLExecute** function executes a prepared SQL statement using the current value of the marker.

Prototype

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

Parameters

Table 13-33 SQLExecute parameters

Parameter	Description
StatementHandle	Statement handle to be executed.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_NEED_DATA** indicates that there are not enough parameters provided to execute the SQL statement.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLExecute** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLFetch

Function

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

Prototype

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

Parameters**Table 13-34** SQLFetch parameters

Parameter	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle .

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLFetch** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLFreeStmt

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

SQLFreeConnect

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

SQLFreeHandle

Function

SQLFreeHandle releases resources associated with a specific environment, connection, or statement handle. It replaces the ODBC 2.x functions: **SQLFreeEnv**, **SQLFreeConnect**, and **SQLFreeStmt**.

Prototype

```
SQLRETURN SQLFreeHandle(SQLSMALLINT HandleType,  
                        SQLHANDLE Handle);
```

Parameters

Table 13-35 SQLFreeHandle parameters

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

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

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

Examples

See [ODBC Development Example](#).

SQLFreeEnv

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

SQLPrepare

Function

SQLPrepare prepares an SQL statement to be executed.

Prototype

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

Parameters

Table 13-36 SQLPrepare parameters

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

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLPrepare** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLGetData

Function

SQLGetData retrieves data for a single column in the current row of the result set. It can be called multiple times to retrieve data of variable lengths.

Prototype

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

Parameters

Table 13-37 SQLGetData parameters

Parameter	Description
StatementHandle	Statement handle, obtained from SQLAllocHandle .
Col_or_Param_Num	Column number of the data to be returned. The columns in the result set are numbered from 1 in ascending order. The number of the bookmark column is 0.
TargetType	Type identifier of the C data type in the TargetValuePtr buffer. If TargetType is SQL_ARD_TYPE , the driver uses the data type of the SQL_DESC_CONCISE_TYPE field in ARD. If TargetType is SQL_C_DEFAULT , the driver selects a default data type according to the source SQL data type.
TargetValuePtr	Output parameter: pointer to the pointer that points to the buffer where the data is located.
BufferLength	Size of the buffer pointed to by TargetValuePtr .
StrLen_or_IndPtr	Output parameter: pointer to the buffer where the length or identifier value is returned.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_NO_DATA** indicates that no result set is returned for the SQL statement.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.
- **SQL_STILL_EXECUTING** indicates that the statement is being executed.

Precautions

If **SQLFetch** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call **SQLGetDiagRec**, set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLGetDiagRec

Function

SQLGetDiagRec returns the current values of multiple fields of a diagnostic record that contains error, warning, and status information.

Prototype

```
SQLRETURN SQLGetDiagRec(SQLSMALLINT HandleType,  
                        SQLHANDLE Handle,  
                        SQLSMALLINT RecNumber,  
                        SQLCHAR *SQLState,  
                        SQLINTEGER *NativeErrorPtr,  
                        SQLCHAR *MessageText,  
                        SQLSMALLINT BufferLength,  
                        SQLSMALLINT *TextLengthPtr);
```

Parameters

Table 13-38 SQLGetDiagRec parameters

Parameter	Description
HandleType	Handle type identifier that describes the handle type required for diagnosis. The value must be one of the following: • SQL_HANDLE_ENV • SQL_HANDLE_DBC • SQL_HANDLE_STMT • SQL_HANDLE_DESC
Handle	Handle of the diagnosis data structure. Its type is indicated by HandleType. If HandleType is SQL_HANDLE_ENV , Handle may be 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 is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.

- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

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

- **SQL_SUCCESS**: The function successfully returns diagnostic information.
- **SQL_SUCCESS_WITH_INFO**: The ***MessageText** buffer is too small to hold the requested diagnostic message and no diagnostic records are generated.
- **SQL_INVALID_HANDLE**: The handle specified by **HandType** and **Handle** is invalid.
- **SQL_ERROR**: **RecNumber** is smaller than or equal to zero, or **BufferLength** is smaller than zero.

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

Table 13-39 SQLSTATE values

SQLSTATE	Error	Description
HY000	General error	An error occurred for which there is no specific SQLSTATE .
HY001	Memory allocation error	The driver is unable to allocate memory required to support execution or completion of the function.
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 connection times out before the data source responds to the request.
IM001	Function not supported by the driver	A function that is not supported by the StatementHandle driver is called.

Examples

See [ODBC Development Example](#).

SQLSetConnectAttr

Function

SQLSetConnectAttr sets connection attributes.

Prototype

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

Parameters

Table 13-40 SQLSetConnectAttr parameters

Parameter	Description
ConnectionHandle	Connection handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the value of Attribute . ValuePtr depends on the value of Attribute and can be a 32-bit unsigned integer value or a null-terminated string. If ValuePtr parameter is driver-specific value, it may be signed integer.
StringLength	If ValuePtr points to a string or a binary buffer, this parameter should be the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLSetConnectAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_DBC** and **ConnectionHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLSetEnvAttr

Function

SQLSetEnvAttr sets environment attributes.

Prototype

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

Parameters

Table 13-41 SQLSetEnvAttr parameters

Parameter	Description
EnvironmentHandle	Environment handle.
Attribute	Environment attribute to be set. Its value must be one of the following: • SQL_ATTR_ODBC_VERSION: ODBC version • SQL_CONNECTION_POOLING: connection pool attribute • SQL_OUTPUT_NTS: string type returned by the driver
ValuePtr	Pointer to the value of Attribute . ValuePtr depends on the value of Attribute and can be a 32-bit integer value or a null-terminated string.
StringLength	If ValuePtr points to a string or a binary buffer, this parameter should be the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLSetEnvAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_ENV** and **EnvironmentHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

SQLSetStmtAttr

Function

SQLSetStmtAttr sets attributes related to a statement.

Prototype

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

Parameters

Table 13-42 SQLSetStmtAttr parameters

Parameter	Description
StatementHandle	Statement handle.
Attribute	Attribute to set.
ValuePtr	Pointer to the value of Attribute . ValuePtr depends on the value of Attribute and can be a 32-bit unsigned integer value or a pointer to a null-terminated string, a binary buffer, and a driver-specified value. If ValuePtr parameter is driver-specific value, it may be signed integer.
StringLength	If ValuePtr points to a string or a binary buffer, this parameter should be the length of *ValuePtr . If ValuePtr points to an integer, StringLength is ignored.

Return values

- **SQL_SUCCESS** indicates that the call is successful.
- **SQL_SUCCESS_WITH_INFO** indicates warning information.
- **SQL_ERROR** indicates major errors, such as memory allocation and connection setup failures.
- **SQL_INVALID_HANDLE** indicates that invalid handles were called. Values returned by other APIs are similar to the values returned by the API you have used.

Precautions

If **SQLSetStmtAttr** returns **SQL_ERROR** or **SQL_SUCCESS_WITH_INFO**, the application can then call [SQLGetDiagRec](#), set **HandleType** and **Handle** to **SQL_HANDLE_STMT** and **StatementHandle**, and obtain the **SQLSTATE** value. This value can be used to get more information about the function call.

Examples

See [ODBC Development Example](#).

14 DWS Hot and Cold Data Management

Introduction to Hot and Cold Data

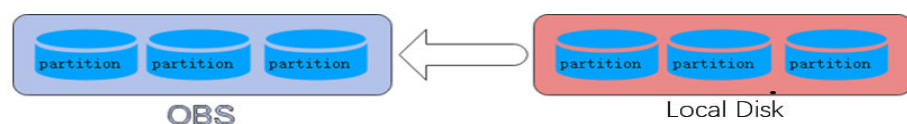
In massive big data scenarios, as services and data volume increase, data storage and consumption increase. The need for data may vary in different time periods, therefore, data is managed in a hierarchical manner, improving data analysis performance and reducing service costs.

For example, in a network traffic analysis system, users may be interested in security events and network access in the last month, but seldom pay attention to data generated several months ago. In such scenarios, data can be classified into hot data and cold data based on time periods.

Hot and cold data is classified based on the data access frequency and update frequency.

- Hot data: Data that is frequently accessed and updated, has a high probability of being invoked in the future, and has high requirements on access response time.
- Cold: Data that cannot be updated or is seldom updated, seldom accessed, and has low requirements on response time.

You can define cold and hot management tables to switch cold data that meets the specified rules to OBS for storage. Cold and hot data can be automatically determined and migrated by partition.



Hot and Cold Data Migration

When data is written to DWS column-store tables, the data is first stored in hot partitions. If there is a large amount of data in the partitions, the data that meets the cold data rules can be manually or automatically migrated to OBS for storage.

The metadata, description tables, and indexes of the migrated cold data are stored locally to ensure the read performance.

Cold/Hot Switchover Policies

Currently, the hot and cold partitions can be switched based on LMT (Last Modify Time) and HPN (Hot Partition Number) policies. LMT indicates that the switchover is performed based on the last update time of the partition, and HPN indicates that the switchover is performed based on the number of reserved hot partitions.

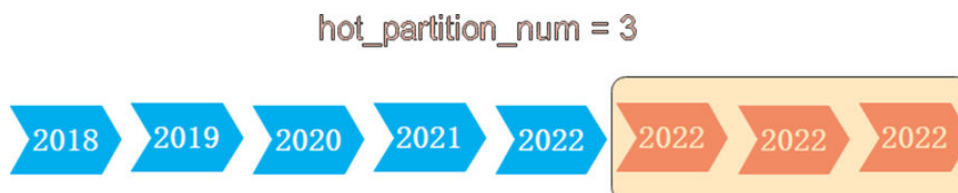
- **LMT:** Switch the hot partition data that is not updated in the last *[day]* days to the OBS tablespace as cold partition data. *[day]* is an integer ranging from 0 to 36500, in days.

In the following figure, *day* is set to 2, indicating that the partitions modified in the last two days are retained as the hot partitions, while the rest is retained as the cold partitions. Assume that the current time is April 30. The delete operation is performed on the partition **[4-26]** on April 30, and the insert operation is performed on the partition **[4-27]** on April 29. Therefore, partitions **[4-26]** **[4-27]** **[4-29]** **[4-30]** are retained as hot partitions.



- **HPN:** indicates the number of hot partitions to be reserved. The partitions are sequenced based on partition sequence IDs. The sequence ID of a partition is a built-in sequence number generated based on the partition boundary values and is not shown. For a range partition, a larger boundary value indicates a larger sequence ID. For a list partition, a larger maximum enumerated value of the partition boundary indicates a larger sequence ID. During the cold and hot switchover, data needs to be migrated to OBS. HPN is an integer ranging from 0 to 1600. If HPN is set to 0, hot partitions are not reserved. During a cold/hot switchover, all partitions with data are converted to cold partitions and stored on OBS.

In the following figure, HPN is set to 3, indicating that the last three partitions with data are retained as the hot partitions with the rest as the cold partitions during hot and cold partition switchover.



Hot and cold data management supports the following functions:

- Supports DML operations on cold and hot tables, such as **INSERT**, **COPY**, **DELETE**, **UPDATE**, and **SELECT**.
- Supports DCL operations such as permission management on cold and hot tables.

- Supports ANALYZE, VACUUM, MERGE INTO, and PARTITION operations on cold and hot tables.
- Supports common column-store partitioned tables to be upgraded to hot and cold data tables.
- Supports upgrade, scale-out, scale-in, and redistribution operations on tables with cold and hot data management enabled.
- Supports conversion between cold and hot partitions. This function is supported only in 8.3.0 or later.

Restrictions on Hot and Cold Data Management

- Currently, cold and hot tables support only column-store partitioned tables of version 2.0. Foreign tables do not support cold and hot partitions.
- If you insert data into a cold partition again, the data is directly stored in OBS. It does not turn the cold table into a hot table.
- A partition on a DN is either hot or cold. For a partition across DNs, its data on some DNs may be hot, and some may be cold.
- If a table has both cold and hot partitions, the query becomes slow because cold data is stored on OBS and the read/write speed are lower than those of local queries.
- Only the cold and hot switchover policies can be modified. The tablespace of cold data in cold and hot tables cannot be modified.
- Restrictions on partitioning cold and hot tables:
 - Data in cold partitions cannot be exchanged.
 - **MERGE PARTITION** supports only the merge of hot-hot partitions and cold-cold partitions.
 - Partition operations, such as **ADD**, **MERGE**, and **SPLIT**, cannot be performed on an OBS tablespace.
 - Tablespaces of cold and hot table partitions cannot be specified or modified during table creation.
- Cold and hot data switchover is not performed immediately upon conditions are met. Data switchover is performed only after users manually, or through a scheduler, invoke the switchover command. Currently, the automatic scheduling time is 00:00 every day and can be modified.
- Currently, only the LMT and HPN switchover rules are supported.
- Cold and hot data tables do not support physical fine-grained backup and restoration. Only hot data is backed up during physical backup. Cold data on OBS does not change. The backup and restoration does not support file deletion statements, such as **TRUNCATE TABLE** and **DROP TABLE**.

Examples

1. Create column-store cold and hot tables and set the hot data validity period LMT to 100 days.

```
CREATE TABLE lifecycle_table(i int, val text) WITH (ORIENTATION = COLUMN, storage_policy = 'LMT:100')
PARTITION BY RANGE (i)
(
PARTITION P1 VALUES LESS THAN(5),
PARTITION P2 VALUES LESS THAN(10),
```

```
PARTITION P3 VALUES LESS THAN(15),  
PARTITION P8 VALUES LESS THAN(MAXVALUE)  
)ENABLE ROW MOVEMENT;
```

2. Switch cold data to the OBS tablespace.

- Automatic switchover: The scheduler automatically triggers the switchover at 00:00 every day.

The automatic switchover time can be customized. For example, the time can be changed to 06:30 every morning.

```
SELECT * FROM pg_obs_cold_refresh_time('lifecycle_table', '06:30:00');
```

- Manual switchover

Perform the following operations to manually switch a single table:

```
ALTER TABLE lifecycle_table refresh storage;
```

Perform the following operations to switch over all cold and hot tables in batches:

```
SELECT pg_catalog.pg_refresh_storage();
```

3. Convert cold partition data into hot partition data. This function is supported only in 8.3.0 or later.

Convert all cold partitions to hot partitions.

```
SELECT pg_catalog.reload_cold_partition('lifecycle_table');
```

Convert a specified cold partition to a hot partition:

```
SELECT pg_catalog.reload_cold_partition('lifecycle_table', 'cold_partition_name');
```

4. View data distribution in hot and cold tables.

View the data distribution in a single table:

```
SELECT * FROM pg_catalog.pg_lifecycle_table_data_distribute('lifecycle_table');
```

View data distribution in all hot and cold tables.

```
SELECT * FROM pg_catalog.pg_lifecycle_node_data_distribute();
```

5. Delete partitions and partition data from cold and hot tables.

Delete data from a specified partition in a hot and cold table.

```
ALTER TABLE lifecycle_table TRUNCATE PARTITION P1;
```

Delete a specified partition from a hot and cold table.

```
ALTER TABLE lifecycle_table DROP PARTITION P1;
```

15 DWS Resource Monitoring

DWS provides multiple dimensional resource monitoring views to show the real-time and historical resource usage of tasks.

15.1 User Resource Monitoring

In the multi-tenant management framework, you can query the real-time usage of all user resources (including the memory, number of CPU cores, storage space, temporary space, operator spilling space, and I/Os) in real time through the system views [PG_TOTAL_USER_RESOURCE_INFO](#) and [PGXC_TOTAL_USER_RESOURCE_INFO](#) and the function [GS_WLM_USER_RESOURCE_INFO](#). You can also query the system catalog [GS_WLM_USER_RESOURCE_HISTORY](#) and system view [PGXC_WLM_USER_RESOURCE_HISTORY](#) for the historical usage of user resources.

Precautions

- The CPU, I/O, and memory usage of all jobs on fast and slow lanes (simple jobs on fast lanes and complex jobs on slow lanes) can be monitored.
- Currently, the memory and CPU usage of fast track jobs are not controlled. When the fast lane jobs occupy a large number of resources, the used resources may exceed the resource limit.
- In the DN monitoring view, I/O, memory, and CPU display the resource usage and limits of resource pools.
- In the CN monitoring view, I/O, memory, and CPU display the total resource usage and limit of all DN resource pools in the cluster.
- The DN monitoring information is updated every 5 seconds. CNs collect monitoring information from DNs every 5 seconds. Because each instance updates or collects user monitoring information independently, the monitoring information update time on each instance may be different.
- The auxiliary thread automatically invokes the persistence function every 30 seconds to persist user monitoring data. In normal cases, you do not need to do this.

- When there are a large number of users and a large cluster, querying such real-time views will cause network latency due to the real-time communication overhead between CNs and DN.
- Resources are not monitored for an initial administrator.

Procedure

- Query all users' resource quotas and real-time resource usage.

```
SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO;
```

The result view is as follows:

username	used_memory	total_memory	used_cpu	total_cpu	used_space	total_space	used_temp_space	total_temp_space	used_spill_space	total_spill_space	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed	send_speed	recv_speed
perfdm	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	-1
usern	0	17250	0	48	-1	0	0	0	0	0	0	0	0	0	0	0	0	-1
(2 rows)																		

The I/O resource monitoring fields (**read_kbytes**, **write_kbytes**, **read_counts**, **write_counts**, **read_speed**, and **write_speed**) can be available only when the GUC parameter described in [enable_user_metric_persistent](#) is enabled.

For details about each column, see [PG_TOTAL_USER_RESOURCE_INFO](#).

- Query a user's resource quota and real-time resource usage.

```
SELECT * FROM GS_WLM_USER_RESOURCE_INFO('username');
```

The query result is as follows:

userid	used_memory	total_memory	used_cpu	total_cpu	used_space	total_space	used_temp_space	total_temp_space	used_spill_space	total_spill_space	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed	send_speed	recv_speed
16407	18	1655	6	19	13787176	-1	0	0	0	0	0	0	0	0	0	0	0	-1
(1 row)																		

- Query all users' resource quotas and historical resource usage.

```
SELECT * FROM GS_WLM_USER_RESOURCE_HISTORY;
```

The query result is as follows:

username	timestamp	used_memory	total_memory	used_cpu	total_cpu	used_space	total_space	used_temp_space	total_temp_space	used_spill_space	total_spill_space	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed	send_speed	recv_speed
usern	2020-01-08 22:56:06.456855+08	0	17250	0	48	0	0	0	0	0	0	0	0	0	0	0	0	0	5710
userg	2020-01-08 22:56:06.458659+08	0	15525	33.48	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2206	2020-01-08 22:56:06.460252+08	0	13972	33.48	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2206	2020-01-08 22:56:06.460277	0	136106277	-1	17648	23	0	0	0	0	0	0	0	0	0	0	0	0	0

For the system catalog [GS_WLM_USER_RESOURCE_HISTORY](#), data in the [PG_TOTAL_USER_RESOURCE_INFO](#) view is periodically saved to historical tables only when the GUC parameter [enable_user_metric_persistent](#) is enabled.

For details about each column, see [GS_WLM_USER_RESOURCE_HISTORY](#).

15.2 Resource Pool Monitoring

Overview

In the multi-tenant management framework, if queries are associated with resource pools, the resources occupied by the queries are summarized to the associated resource pools. You can query the real-time resource usage of all resource pools in the resource pool monitoring view and query the historical resource usage of resource pools in the resource pool monitoring history table.

The resource pool monitoring data is updated every 5s. However, due to the time difference between CNs and DN, the actual monitoring data update time may be longer than 5s. Generally, the time does not exceed 10s. The resource pool monitoring data is persisted every 30 seconds. The resource pool monitoring logic is basically the same as that of the user resource monitoring. Therefore, the [enable_user_metric_persistent](#) and [user_metric_retention_time](#) parameters are used to control the persistence and aging of resource pool monitoring data, respectively.

Resources monitored by a resource pool include the running and queuing information of fast and slow lane jobs, and CPU, memory, and logical I/O resource monitoring information. The monitoring views and history tables are as follows:

- Real-time monitoring view of resource pools (single CN):
[GS_RESPOOL_RUNTIME_INFO](#)
- Real-time monitoring view of resource pools (all CNs):
[PGXC_RESPOOL_RUNTIME_INFO](#)
- Real-time monitoring view of resource pool resources (single CN):
[GS_RESPOOL_RESOURCE_INFO](#)
- Real-time monitoring view of resource pool resources (all CNs):
[PGXC_RESPOOL_RESOURCE_INFO](#)
- Historical resource monitoring table of the resource pool (single CN):
[GS_RESPOOL_RESOURCE_HISTORY](#)
- Monitoring view of historical resource pool resources (all CNs):
[PGXC_RESPOOL_RESOURCE_HISTORY](#)

 NOTE

- Resource pool monitoring monitors the CPU, I/O, and memory usage of all jobs on the fast and slow lanes.
- Currently, the memory and CPU usage of fast track jobs are not controlled. When the fast lane jobs occupy a large number of resources, the used resources may exceed the resource limit.
- In the monitoring view of DN resource pools, I/O, memory, and CPU display the resource usage and limits of resource pools.
- In the monitoring view of CN resource pools, I/O, memory, and CPU display the total resource usage and limit of all DN resource pools in the cluster.
- Resource pool monitoring information on DNs is updated every 5 seconds. CNs collect resource pool monitoring information from DNs every 5 seconds. Because each instance updates or collects resource pool monitoring information independently, the monitoring information update time on each instance may be different.
- The auxiliary thread automatically invokes the persistence function every 30 seconds to persist the resource pool monitoring data. In normal cases, you do not need to do this.

Procedure

- Querying the real-time running status of jobs in a resource pool.
SELECT * FROM GS_RESPOOL_RUNTIME_INFO;

The result view is as follows:

nodegroup	rpname	ref_count	fast_run	fast_wait	slow_run	slow_wait
vc1	p2	10	0	0	0	0
vc2	p3	10	5	5	0	0
vc2	p4	0	0	0	0	0
vc1	default_pool	0	0	0	0	0
vc2	default_pool	0	0	0	0	0
vc1	p1	20	5	5	3	7

(6 rows)

Where,

- ref_count** indicates the number of jobs that reference the current resource pool information. Its value will be retained until the management ends.
 - fast_run** and **slow_run** are load management accounting information. Their values are valid only when **fast_limit** and **slow_limit** are larger than 0.
 - This view is valid only on CNs. The persistence information is stored in **GS_RESPOOL_RESOURCE_HISTORY**.
 - For details about each field, see [GS_RESPOOL_RUNTIME_INFO](#).
- Querying the resource quota and real-time resource usage of a resource pool.
SELECT * FROM GS_RESPOOL_RESOURCE_INFO;

The result view is as follows:

nodegroup	rpname	cgroup	ref_count	fast_run	fast_wait	fast_limit	slow_run	slow_wait	slow_limit	used_cpu	cpu_limit	used_mem	estimate_mem	mem_limit	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed	send_speed	recv_speed
-----+-----																						

- This view is valid on both CNs and DNs. The CPU, memory, and I/O usage on a DN indicates the resource consumption of the DN. The CPU, memory, and I/O usage on a CN is the total resource consumption of all DNs in the cluster.
- estimate_mem** is valid only on CNs under dynamic load management. It displays the estimated memory accounting of the resource pool.
- I/O monitoring information is recorded only when **enable_logical_io_statistics** is enabled.
- For details about each field, see [GS_RESPOOL_RESOURCE_INFO](#).

Querying the resource quota and historical resource usage of a resource pool.

```
SELECT * FROM GS_RESPOOL_RESOURCE_HISTORY ORDER BY timestamp DESC;
```

timestamp	nodegroup			rpname	cgroup		ref_count	fast_run	fast_wait
fast_limit	slow_run	slow_wait	slow_limit	used_cpu	cpu_limit	used_mem	estimate_mem		
mem_limit	read_kbytes	write_kbytes	read_counts	write_counts	read_speed	write_speed			
send_speed	rcv_speed								
2022-03-04 09:41:57.53739+08	vc1	p2	DefaultClass:Rush	10	0	0			
-1	0	10	9.97	48	20	0	11555	0	
2320	0	290	0	474	0	0			
2022-03-04 09:41:57.53739+08	vc1	p1	DefaultClass:Rush	20	5	5			
5	3	7	3	7.98	48	16	768	11555	0
1896	0	237	0	387	0	0			
2022-03-04 09:41:57.53739+08	vc2	default_pool	DefaultClass:Medium	0	0				
0	-1	0	0	-1	0	48	0	0	11555
0	0	0	0	0	0	0	0	11555	0
2022-03-04 09:41:57.53739+08	vc1	default_pool	DefaultClass:Medium	0	0				
0	-1	0	0	-1	0	48	0	0	11555
0	0	0	0	0	0	0	0		
2022-03-04 09:41:57.53739+08	vc2	p4	DefaultClass:Rush	0	0	0			
-1	0	0	10	0	48	0	0	11555	0
0	0	0	0	0	0	0			
2022-03-04 09:41:57.53739+08	vc2	p3	DefaultClass:Rush	10	5	5			
5	0	0	10	4.99	48	11	0	11555	0
0	110	0	180	0	0				
2022-03-04 09:41:27.335234+08	vc2	p3	DefaultClass:Rush	10	5	5			
5	0	0	10	4.98	48	11	0	11555	0
0	107	0	175	0	0				

- 594

- c. For details about each field, see [GS_RESPOOL_RESOURCE_HISTORY](#).

15.3 Monitoring Memory Resources

Monitoring the Memory

DWS allows you to view cluster memory usage on the monitoring panel (DMS). For details, see "Viewing DWS Cluster Monitoring Information on the Monitoring Panel (DMS)" in *Data Warehouse Service (DWS) User Guide*.

DWS provides a view for monitoring the memory usage of the entire cluster.

Query the **pgxc_total_memory_detail** view as a user with the **sysadmin** permission.

```
SELECT * FROM pgxc_total_memory_detail;
```

If the following error message is returned during the query, enable the memory management function.

```
SELECT * FROM pgxc_total_memory_detail;  
ERROR: unsupported view for memory protection feature is disabled.  
CONTEXT: PL/pgSQL function pgxc_total_memory_detail() line 12 at FOR over EXECUTE statement
```

You can set **max_process_memory** on the DWS console to enable memory management. The procedure is as follows:

1. Log in to the DWS console.
2. Choose **Dedicated Clusters** > **Clusters** in the navigation tree on the left.
3. In the cluster list, find the target cluster and click the cluster name to go to the cluster details page.
4. Click the **Parameter Modifications** tab and change the value of **max_process_memory** to a proper one. For details about the modification suggestions, see [max_process_memory](#). After it is done, click **Save**.
5. In the **Modification Preview** dialog box, confirm the modifications and click **Save**. After the modification, restart the cluster for the modification to take effect.

15.4 Instance Resource Monitoring

DWS provides system catalogs for monitoring the resource usage of CNs and DN (including memory, CPU usage, disk I/O, process physical I/O, and process logical I/O), and system catalogs for monitoring the resource usage of the entire cluster.

For details about the system catalog **GS_WLM_INSTANCE_HISTORY**, see [GS_WLM_INSTANCE_HISTORY](#).

NOTE

Data in the system catalog **GS_WLM_INSTANCE_HISTORY** is distributed in corresponding instances. CN monitoring data is stored in the CN instance, and DN monitoring data is stored in the DN instance. The DN has a standby node. When the primary DN is abnormal, the monitoring data of the DN can be restored from the standby node. However, a CN has no standby node. When a CN is abnormal and then restored, the monitoring data of the CN will be lost.

Procedure

- Query the latest resource usage of the current instance.

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY ORDER BY TIMESTAMP DESC;
```

The query result is as follows:

instancename	timestamp	used_cpu	free_mem	used_mem	io_await	io_util	disk_read	disk_write	process_read	process_write	logical_read	logical_write	read_counts	write_counts
dn_6015_6016	2022-01-10 17:29:17.329495+08	0	14570	8982	662.923	99.9601	697666	93655.5	183104	30082	285659	30079	357717	37667
dn_6015_6016	2022-01-10 17:29:07.312049+08	0	14578	8974	883.102	99.9801	756228	81417.4	189722	30786	285681	30780	358103	38584
dn_6015_6016	2022-01-10 17:28:57.284472+08	0	14583	8969	727.135	99.9801	648581	88799.6	177120	31176	252161	31175	316085	39079
dn_6015_6016	2022-01-10 17:28:47.256613+08	0	14591	8961	679.534	100.08	655360	169962	179404	30424	242002	30422	303351	38136

- Query the resource usage of the current instance during a specified period.

```
SELECT * FROM GS_WLM_INSTANCE_HISTORY WHERE TIMESTAMP > '2022-01-10' AND TIMESTAMP < '2020-01-11' ORDER BY TIMESTAMP DESC;
```

The query result is as follows:

instancename	timestamp	used_cpu	free_mem	used_mem	io_await	io_util	disk_read	disk_write	process_read	process_write	logical_read	logical_write	read_counts	write_counts
dn_6015_6016	2022-01-10 17:29:17.329495+08	0	14570	8982	662.923	99.9601	697666	93655.5	183104	30082	285659	30079	357717	37667
dn_6015_6016	2022-01-10 17:29:07.312049+08	0	14578	8974	883.102	99.9801	756228	81417.4	189722	30786	285681	30780	358103	38584
dn_6015_6016	2022-01-10 17:28:57.284472+08	0	14583	8969	727.135	99.9801	648581	88799.6	177120	31176	252161	31175	316085	39079
dn_6015_6016	2022-01-10 17:28:47.256613+08	0	14591	8961	679.534	100.08	655360	169962	179404	30424	242002	30422	303351	38136

- To query the latest resource usage of a cluster, you can invoke the **pgxc_get_wlm_current_instance_info** stored procedure on the CN.

```
SELECT * FROM pgxc_get_wlm_current_instance_info('ALL');
```

The query result is as follows:

instancename	timestamp	used_cpu	free_mem	used_mem	io_await	io_util	disk_read	disk_write	process_read	process_write	logical_read	logical_write	read_counts	write_counts
coordinator2	2020-01-14 21:58:29.290894+08	0	12010	278	16.0445	7.19561	184.431	27959.3	0	10	0	0	0	0
coordinator3	2020-01-14 21:58:27.567655+08	0	12000	288	.964557	3.40659	332.468	3375.02	26	13	0	0	0	0
datanode1	2020-01-14 21:58:23.900321+08	0	11899	389	1.17296	3.25	329.6	2870.4	28	8	13	3	18	6
datanode2	2020-01-14 21:58:32.832989+08	0	11904	384	17.948	8.52148	214.186	25894.1	28	10	13	3	18	6
datanode3	2020-01-14 21:58:24.826694+08	0	11894	394	1.16088	3.15	328	2868.8	25	10	13	3	18	6
coordinator1	2020-01-14 21:58:33.367649+08	0	11988	300	9.53286	10.05	43.2	55232	0	0	0	0	0	0
coordinator1	2020-01-14 21:58:23.216645+08	0	11988	300	1.17085	3.21182	324.729	2831.13	8	13	0	0	0	0

(7 rows)

- To query the historical resource usage of a cluster, you can call the **pgxc_get_wlm_history_instance_info** stored procedure function on the CN.

```
SELECT * FROM pgxc_get_wlm_history_instance_info('ALL', '2020-01-14 21:00:00', '2020-01-14 22:00:00', 3);
```

The query result is as follows:

```

instance | timestamp | used_cpu | free_mem | used_mem | io_await | io_util |
disk_read | disk_write | process_read | process_write | logical_read | logical_write | read_counts |
write_counts
-----+-----+-----+-----+-----+-----+-----+
coordinator2 | 2020-01-14 21:50:49.778902+08 | 0 | 12020 | 268 | .127371 | .789211 |
15.984 | 3994.41 | 0 | 0 | 0 | 0 | 0 |
coordinator2 | 2020-01-14 21:53:49.043646+08 | 0 | 12018 | 270 | 30.2902 | 8.65404 |
276.77 | 16741.8 | 3 | 1 | 0 | 0 | 0 |
coordinator2 | 2020-01-14 21:57:09.202654+08 | 0 | 12018 | 270 | .16051 | .979021 |
59.9401 | 5596 | 0 | 0 | 0 | 0 | 0 |
coordinator3 | 2020-01-14 21:38:48.948646+08 | 0 | 12012 | 276 | .0769231 | .00999001 |
0 | 35.1648 | 0 | 1 | 0 | 0 | 0 |
coordinator3 | 2020-01-14 21:40:29.061178+08 | 0 | 12012 | 276 | .118421 | .0199601 |
0 | 970.858 | 0 | 0 | 0 | 0 | 0 |
coordinator3 | 2020-01-14 21:50:19.612777+08 | 0 | 12010 | 278 | 24.411 | 11.7665 |
8.78244 | 44641.1 | 0 | 0 | 0 | 0 | 0 |
datanode1 | 2020-01-14 21:49:42.758649+08 | 0 | 11909 | 379 | .798776 | 8.02 |
51.2 | 20924.8 | 0 | 0 | 0 | 0 | 0 |
datanode1 | 2020-01-14 21:49:52.760188+08 | 0 | 11909 | 379 | 23.8972 | 14.1 |
0 | 74760 | 0 | 0 | 0 | 0 | 0 |
datanode1 | 2020-01-14 21:50:22.769226+08 | 0 | 11909 | 379 | 39.5868 | 7.4 | 0
19760.8 | 0 | 0 | 0 | 0 | 0 | 0 |
datanode2 | 2020-01-14 21:58:02.826185+08 | 0 | 11905 | 383 | .351648 | .32 |
20.8 | 504.8 | 0 | 0 | 0 | 0 | 0 |
datanode2 | 2020-01-14 21:56:42.80793+08 | 0 | 11906 | 382 | .559748 | .04 | 0
326.4 | 0 | 0 | 0 | 0 | 0 | 0 |
datanode2 | 2020-01-14 21:45:21.632407+08 | 0 | 11901 | 387 | 12.1313 | 4.55544 |
3.1968 | 45177.2 | 0 | 0 | 0 | 0 | 0 |
datanode3 | 2020-01-14 21:58:14.823317+08 | 0 | 11898 | 390 | .378205 | .99 |
48 | 23353.6 | 0 | 0 | 0 | 0 | 0 |
datanode3 | 2020-01-14 21:47:50.665028+08 | 0 | 11901 | 387 | 1.07494 | 1.19 |
0 | 15506.4 | 0 | 0 | 0 | 0 | 0 |
datanode3 | 2020-01-14 21:51:21.720117+08 | 0 | 11903 | 385 | 10.2795 | 3.11 |
0 | 11031.2 | 0 | 0 | 0 | 0 | 0 |
coordinator1 | 2020-01-14 21:42:59.121945+08 | 0 | 12020 | 268 | .0857143 | .0699301 |
0 | 6579.02 | 0 | 0 | 0 | 0 | 0 |
coordinator1 | 2020-01-14 21:41:49.042646+08 | 0 | 12020 | 268 | 20.9039 | 11.3786 |
6042.76 | 57903.7 | 0 | 0 | 0 | 0 | 0 |
coordinator1 | 2020-01-14 21:41:09.007652+08 | 0 | 12020 | 268 | .0446429 | .03996 |
0 | 1109.29 | 0 | 0 | 0 | 0 | 0 |
(18 rows)

```

15.5 Real-time Top SQL

You can query real-time Top SQL in real-time resource monitoring views at different levels. The real-time resource monitoring view records the resource usage (including memory, data spilled to disks, and CPU time) and performance alarm information during job running.

The following table describes the external interfaces of the real-time views.

Table 15-1 Real-time resource monitoring views

Level	Monitored Node	View
Query level/perf level	Current CN	GS_WLM_SESSION_STATISTICS
	All CNs	PGXC_WLM_SESSION_STATISTICS
operator level	Current CN	GS_WLM_OPERATOR_STATISTICS

Level	Monitored Node	View
	All CNs	PGXC_WLM_OPERATOR_STATISTICS

 NOTE

- The view level is determined by the resource monitoring level, that is, the [resource_track_level](#) configuration.
- The perf and operator levels affect the values of the **query_plan** and **warning** columns in [GS_WLM_SESSION_STATISTICS](#) or [PGXC_WLM_SESSION_INFO](#). For details, see [SQL Self-Diagnosis](#).
- Prefixes **gs** and **pgxc** indicate views showing single CN information and those showing cluster information, respectively. Common users can log in to a CN in the cluster to query only views with the **gs** prefix.
- When you query this type of views, there will be network latency, because the views obtain resource usage in real time.
- If an instance fault occurs, some Top SQL statement information may fail to be recorded in real-time resource monitoring views.
- Top SQL statements are recorded in real-time resource monitoring views as follows:
 - Special DDL statements, such as **SET**, **RESET**, **SHOW**, **ALTER SESSION SET**, and **SET CONSTRAINTS**, are not recorded.
 - DDL statements, such as **CREATE**, **ALTER**, **DROP**, **GRANT**, **REVOKE**, and **VACUUM**, are recorded.
 - DML statements are recorded, including:
 - the execution of **SELECT**, **INSERT**, **UPDATE**, and **DELETE**
 - the execution of **EXPLAIN ANALYZE** and **EXPLAIN PERFORMANCE**
 - the use of the query-level or perf-level views
 - The entry statements for invoking functions and stored procedures are recorded. When the GUC parameter [enable_track_record_subsql](#) is enabled, some internal statements (except the **DECLARE** definition statement) of a stored procedure can be recorded. Only the internal statements delivered to DNs for execution are recorded, and the remaining internal statements are filtered out.
 - The anonymous block statement is recorded. When the GUC parameter [enable_track_record_subsql](#) is enabled, some internal statements of an anonymous block can be recorded. Only the internal statements delivered to DNs for execution are recorded, and the remaining internal statements are filtered out.
 - The cursor statements are recorded. If a cursor does not read data from the cache but triggers the condition for delivering the statement to a DN for execution, the cursor statement is recorded and the statement and execution plan are enhanced. However, if the cursor reads data from the cache, the cursor statement is not recorded. When a cursor statement is used in an anonymous block or function and the cursor reads a large amount of data from a DN but is not fully used, the monitoring information about the cursor on the DN cannot be recorded due to the current architecture limitation. The **With Hold** cursor syntax has a special execution logic. It executes queries during transaction committing. If a statement execution error is reported during this period of time, the **aborted** status of the job cannot be recorded in the TopSQL history table.
 - Jobs in a redistribution process are not monitored.
 - The parameters of a statement with placeholders executed by JDBC are generally specified. However, if the length of the parameter and the original statement exceeds 64 KB, the parameter is not recorded. If the statement is a lightweight statement, it is directly delivered to the DN for execution and the parameter is not recorded.
 - In cluster 8.1.3 and later versions, the TopSQL monitoring at the query and perf levels does not affect the query performance. The default value of the GUC parameter **resource_track_cost** for resource monitoring of statements has been changed to **0**. When you query the TopSQL real-time monitoring view, by default, all statements that are being executed are displayed.
 - In 8.1.3 and later versions, if the GUC parameter [enable_track_record_subsql](#) for querying the TopSQL monitoring view is enabled, regardless of whether the

substatement monitoring function is enabled in the service statements, you can view the substatement running information in the TopSQL monitoring view.

- You are advised not to fully enable substatement monitoring in stored procedures, that is, **enable_track_record_subsql**, in the 8.1.3 cluster version. Because the substatements cannot be filtered by time, fully enabling substatement monitoring may record too many substatements. As a result, archived monitoring tables occupy a large amount of disk space. In the 8.1.3 cluster version, you are advised to enable only the parameters in the corresponding session when querying real-time monitoring information or locating and analyzing some stored procedures. Starting from cluster versions 8.2.1 and later, a new customizable GUC parameter **resource_track_subsql_duration** is added. By default, it is set to 180 seconds. This parameter allows you to filter and archive substatements based on their execution time.
- Due to specification restrictions, the records of the main statements that are not written to disks in the TopSQL history table are delayed. The records are displayed in the TopSQL history table only when the job is delivered next time.
- In the 8.2.1.200 cluster version, operator_realtime-level top SQL runtime monitoring is added to provide operator-level real-time monitoring. After operator_realtime-level monitoring is enabled, you can query the execution plan and detailed execution information of statements. When you query the operator-level real-time monitoring view of top SQL statements, by default, all statements that are being executed are displayed. However, in stored procedure and cursor scenarios, operator-level real-time monitoring information cannot be displayed. Querying historical operator information is not supported in this version.
- operator_realtime-level TopSQL runtime monitoring is not supported for lightweight CN statements and stored procedures. In addition, due to the high execution speed of operators, the display of the operator information may lag behind.
- The **spill_size** field at the query level (job monitoring) and operator level (operator monitoring) varies due to the statistical dimension. The spill size at the query level is the statement files spilled to disks, and the spill size at the operator level is the read and write I/O volume of a specific operator at the logical layer.
- When the GUC parameter **enable_stream_operator** is set to off, the displayed operator execution information may be inaccurate.

Prerequisites

- The GUC parameter **enable_resource_track** is set to **on**. The default value is **on**.
- The GUC parameter **resource_track_level** is set to **query**, **perf**, **operator_realtime**, or **operator**. The default value is **query**.
- Job monitoring rules are as follows:
 - Jobs whose execution cost estimated by the optimizer is greater than or equal to **resource_track_cost**.
- If the Cgroups function is properly loaded, you can run the **gs_cgroup -P** command to view information about Cgroups.
- The GUC parameter **enable_track_record_subsql** specifies whether to record internal statements of a stored procedure or anonymous block.

In the preceding prerequisites, **enable_resource_track** is a system-level parameter that specifies whether to enable resource monitoring. **resource_track_level** is a session-level parameter. You can set the resource monitoring level of a session as needed. The following table describes the values of the two parameters.

Table 15-2 Setting the resource monitoring level to collect statistics

enable_resource_track	resource_track_level	Query-Level Information	Operator-Level Information
on(default)	none	Not collected	Not collected
	query(default)	Collected	Not collected
	perf	Collected	Not collected
	operator	Collected	Collected
on(default)	operator_realtime	Collected	Real-time operator monitoring
off	none/query/operator	Not collected	Not collected

Procedure

Step 1 Query for the real-time CPU information in the **gs_session_cpu_statistics** view.

```
SELECT * FROM gs_session_cpu_statistics;
```

Step 2 Query for the real-time memory information in the **gs_session_memory_statistics** view.

```
SELECT * FROM gs_session_memory_statistics;
```

Step 3 Query for the real-time resource information about the current CN in the **gs_wlm_session_statistics** view.

```
SELECT * FROM gs_wlm_session_statistics;
```

Step 4 Query for the real-time resource information about all CNs in the **pgxc_wlm_session_statistics** view.

```
SELECT * FROM pgxc_wlm_session_statistics;
```

Step 5 Query for the real-time resource information about job operators on the current CN in the **gs_wlm_operator_statistics** view.

```
SELECT * FROM gs_wlm_operator_statistics;
```

Step 6 Query for the real-time resource information about job operators on all CNs in the **pgxc_wlm_operator_statistics** view.

```
SELECT * FROM pgxc_wlm_operator_statistics;
```

Step 7 Query for the load management information about the jobs executed by the current user in the **PG_SESSION_WLMSTAT** view.

```
SELECT * FROM pg_session_wlmstat;
```

Step 8 Query the job execution status of the current user on each CN in the **pgxc_wlm_workload_records** view (this view is available when the dynamic load function is enabled, that is, **enable_dynamic_workload** is set to **on**).

```
SELECT * FROM pgxc_wlm_workload_records;
```

----End

15.6 Historical Top SQL

You can query historical Top SQL in historical resource monitoring views. The historical resource monitoring view records the resource usage (including memory, data spilled to disks, and CPU time), running status (including errors, termination, and exceptions), and performance alarm information when a job is complete. For queries that abnormally terminate due to FATAL or PANIC errors, their status is displayed as **aborted** and no detailed information is recorded. Status information about query parsing in the optimization phase cannot be monitored.

The following table describes the external interfaces of the historical views.

Level	Monitored Node	View	
Query level/perf level (recommended)	Current CN	History (Internal dump interface. Only statements that have ended in the last three minutes are displayed.)	GS_WLM_SESSION_HISTORY
		History (all statements)	GS_WLM_SESSION_INFO
	All CNs	History (Internal dump interface. Only statements that have ended in the last three minutes are displayed.)	PGXC_WLM_SESSION_HISTORY
		History (all statements)	PGXC_WLM_SESSION_INFO
Operator level	Current CN	History (Only statements that have ended in the last three minutes are displayed.)	GS_WLM_OPERATOR_HISTORY
		History (internal dump interface, all statements)	GS_WLM_OPERATOR_INFO
	All CNs	History (Only statements that have ended in the last three minutes are displayed.)	PGXC_WLM_OPERATOR_HISTORY
		History (internal dump interface, all statements)	PGXC_WLM_OPERATOR_INFO

 NOTE

- The view level is determined by the resource monitoring level, that is, the [resource_track_level](#) configuration.
- The perf and operator levels affect the values of the **query_plan** and **warning** columns in [GS_WLM_SESSION_STATISTICS](#) or [PGXC_WLM_SESSION_INFO](#). For details, see [SQL Self-Diagnosis](#).
- Prefixes **gs** and **pgxc** indicate views showing single CN information and those showing cluster information, respectively. Common users can log in to a CN in the cluster to query only views with the **gs** prefix.
- If instance fault occurs, some SQL statement information may fail to be recorded in historical resource monitoring views.
- In some abnormal cases, the status information column in the historical Top SQL may be displayed as **unknown**. The recorded monitoring information may be inaccurate.
- The SQL statements that can be recorded in historical resource monitoring views are the same as those recorded in real-time resource monitoring views. For details, see [SQL statements recorded in real-time resource monitoring views](#).
- Historical top SQL statements are recorded only when the GUC parameter [enable_resource_record](#) is enabled.
- You can query historical Top SQL queries and operator-level data only through the PostgreSQL database.
- Historical Top SQL focuses on locating and demarcating query performance problems. It is not used for auditing or recording syntax analysis error statements.
- In 8.2.1 and later cluster versions, the **resource_track_subsql_duration** parameter (default value: 180s) is added to filter out substatements in the stored procedure whose execution time is less than the value of this parameter and archive only substatements whose execution time is greater than the value of this parameter. In 8.2.1 and later versions, the default value of **enable_track_record_subsql** is changed from **off** to **on**, which means substatements in stored procedures are recorded by default. If a substatement is recorded, it must meet the following conditions:
 - In the session where the statement is, the **enable_track_record_subsql** parameter is enabled.
 - The substatement must be pushed down to DN for execution. (To prevent TopSQL from recording too many substatements, substatements that are not pushed down to DN will be filtered out.)
 - The execution time of the substatement exceeds the value of **resource_track_subsql_duration** in the session.
- By default, the History view queries statements that end in the last 3 minutes. It does this by querying tables. It is actually a temporary view for performance considerations. Since the 8.1.3 cluster version, the real-time monitoring and archiving functions of the TopSQL monitoring have been greatly improved and no performance considerations are needed. Therefore, you are not advised to use the History view.
- In 8.1.3 and later versions, the TopSQL real-time monitoring has no impact on statement performance. You can set the GUC **parameter resource_track_cost** to **0** to monitor the running information of all statements. The statement archiving in the TopSQL history monitoring also has no impact on statement performance. However, when the TPS is high, the following factors need to be considered:
 - Record the disk overhead of all statements. You can estimate the disk space required for archiving a statement as 8 KB, calculate the space usage based on the peak TPS, and adjust the values of **resource_track_duration** and **resource_track_subsql_duration**.
 - For memory overhead for caching all statements, you can estimate the memory size required for archiving a statement as 16 KB, and the interval for archiving statements in batches as 5 seconds, then calculate the required peak memory size based on the peak service TPS. The calculation method is as follows: 5 seconds x TPS x 16 KB. The value of **session_history_memory** GUC (default value: 100 MB)

must be greater than the calculation result to ensure that all statements can be recorded.

Prerequisites

- The GUC parameter `enable_resource_track` is set to **on**. The default value is **on**.
- The GUC parameter `resource_track_level` is set to **query**, **perf**, or **operator**. The default value is **query**. For details, see [Table 15-2](#).
- The GUC parameter `enable_resource_record` is set to **on**. The default value is **on**.
- The GUC parameter `resource_track_duration` is less than the sum of the job execution time and queuing time (**60s** by default).
- The GUC parameter `enable_track_record_subsql` specifies whether to record internal statements of a stored procedure or anonymous block. The default value is **on**.
- The value of `resource_track_subsql_duration` is less than the execution time of the internal statement in the stored procedure (**180s** by default).
- Jobs whose sum of the job execution time and queuing time recorded in the real-time resource monitoring view (see [Table 15-1](#)) is no less than the value of `resource_track_duration` are monitored.
- If the Cgroups function is properly loaded, you can run the `gs_cgroup -P` command to view information about Cgroups.

Procedure

Step 1 Query the load records of the current CN after its latest job is complete in the `gs_wlm_session_history` view.

```
SELECT * FROM gs_wlm_session_history;
```

Step 2 Query the load records of all the CNs after their latest job are complete in the `pgxc_wlm_session_history` view.

```
SELECT * FROM pgxc_wlm_session_history;
```

Step 3 Query the load records of the current CN through the `gs_wlm_session_info` table after the task is complete. To query the historical records successfully, set `enable_resource_record` to **on**.

```
SELECT * FROM gs_wlm_session_info;
```

- Show the 10 queries that consume the most memory (You can specify a query period.):

```
SELECT * FROM gs_wlm_session_info order by max_peak_memory desc limit 10;  
SELECT * FROM gs_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and finish_time  
<='2022-05-15 23:30:00' order by max_peak_memory desc limit 10;
```

- Show the 10 queries consuming the most CPU resources:

```
SELECT * FROM gs_wlm_session_info order by total_cpu_time desc limit 10;  
SELECT * FROM gs_wlm_session_info WHERE start_time >= '2022-05-15 21:00:00' and finish_time  
<='2022-05-15 23:30:00' order by total_cpu_time desc limit 10;
```

Step 4 Query for the load records of all the CNs after their jobs are complete in the `pgxc_wlm_session_info` view. To query the historical records successfully, set `enable_resource_record` to **on**.

```
SELECT * FROM pgxc_wlm_session_info;
```

- Showing the 10 queries on which the CN spends the most time:

```
SELECT * FROM pgxc_wlm_session_info order by duration desc limit 10;
```

- Query the execution information about a query statement that has been executed. For example, query the execution information about the statement whose **queryid** is **76561193695026478**.

```
SELECT * FROM pgxc_wlm_session_info where queryid = '76561193695026478';
```

Step 5 Use the **pgxc_get_wlm_session_info_bytime** function to filter and query the **pgxc_wlm_session_info** view. To query the historical records successfully, set **enable_resource_record** to **on**. You are advised to use this function if the view contains a large number of records.

NOTE

A DWS cluster uses the UTC time by default, which has an 8-hour time difference with the system time. Before queries, ensure that the database time is the same as the system time.

- Return the queries started between **2019-09-10 15:30:00** and **2019-09-10 15:35:00** on all CNs. For each CN, a maximum of 10 queries will be returned.

```
SELECT * FROM pgxc_get_wlm_session_info_bytime('start_time', '2019-09-10 15:30:00', '2019-09-10 15:35:00', 10);
```

- Return the queries ended between **2019-09-10 15:30:00** and **2019-09-10 15:35:00** on all CNs. For each CN, a maximum of 10 queries will be returned.

```
SELECT * FROM pgxc_get_wlm_session_info_bytime('finish_time', '2019-09-10 15:30:00', '2019-09-10 15:35:00', 10);
```

Step 6 Query the recent resource information of the job operators on the current CN in the **gs_wlm_operator_history** view. Ensure that **resource_track_level** is set to **operator**.

```
SELECT * FROM gs_wlm_operator_history;
```

Step 7 Query the recent resource information of the job operators on all the CNs in the **pgxc_wlm_operator_history** view. Ensure that **resource_track_level** is set to **operator**.

```
SELECT * FROM pgxc_wlm_operator_history;
```

Step 8 Query the recent resource information of the job operators on the current CN in the **gs_wlm_operator_info** view. Ensure that **resource_track_level** is set to **operator** and **enable_resource_record** to **on**.

```
SELECT * FROM gs_wlm_operator_info;
```

Step 9 Query for the historical resource information of job operators on all the CNs in the **pgxc_wlm_operator_info** view. Ensure that **resource_track_level** is set to **operator** and **enable_resource_record** to **on**.

```
SELECT * FROM pgxc_wlm_operator_info;
```

----End

 NOTE

- The number of data records that can be retained in the memory is limited due to the preset memory limit. After the real-time query is complete, the data records are imported to historical views. For a query-level view, when the number of queries to be recorded exceeds the upper limit allowed by the memory, the current query cannot be recorded and the next query is performed based on a new rule. On each CN, the memory usage of the query-level historical view is recorded (100 MB by default). You can query the data in the [PG_TOTAL_MEMORY_DETAIL](#) view.
- For operator-level views, whether a record can be stored depends on the upper limit allowed by the memory at that time point. If the number of plan nodes plus the number of records in the memory exceeds the upper limit, the record cannot be stored. On each CN, the maximum numbers of real-time and historical operator-level records that can be stored in the memory are **max_oper_realt_num** (set to **56987** by default) and **max_oper_hist_num** (set to **113975** by default), respectively. The average number of plan nodes of a query is **num_plan_node**. Maximum number of concurrent tasks allowed by real-time views on each CN is: **num_realt_active = max_oper_realt_num / num_plan_node**. Maximum number of concurrent tasks allowed by historical views on each CN is: **num_hist_active = max_oper_hist_num / (180 / run_time) / num_plan_node**.
- In high concurrency, ensure that the number of queries to be recorded does not exceed the maximum values set for query- and operator-level views. You can modify the memory of the historical query view by configuring the [session_history_memory](#) parameter. The memory size increases in direct proportion to the maximum number of queries that can be recorded.

15.7 Example for Querying for Top SQLs

In this section, TPC-DS sample data is used as an example to describe how to query [Real-time Top SQL](#) and [Historical Top SQL](#).

Configuring Cluster Parameters

To query for historical or archived resource monitoring information about jobs of top SQLs, you need to set related GUC parameters first. The procedure is as follows:

1. Log in to the DWS console.
2. Choose **Dedicated Clusters** > **Clusters** in the navigation tree on the left.
3. In the cluster list, find the target cluster and click the cluster name to go to the cluster details page.
4. Click the **Parameter Modifications** tab, change the value of **resource_track_duration**, and click **Save**.
5. In the **Modification Preview** dialog box, confirm the modification and click **Save**.

 NOTE

If **enable_resource_record** is set to **on**, storage space expansion may occur and thereby slightly affects the performance. Therefore, set it to **off** if record archiving is unnecessary.

Example for Querying for Top SQLs

The TPC-DS sample data is used as an example.

- Step 1** Open the SQL client tool and connect to your database.
- Step 2** Run the **EXPLAIN** statement to query for the estimated cost of the SQL statement to be executed to determine whether resources of the SQL statement will be monitored.

By default, only resources of a query whose execution cost is greater than the value of [resource_track_cost](#) are monitored and can be queried by users.

For example, run the following statements to query for the estimated execution cost of the SQL statement:

```
SET CURRENT_SCHEMA = tpcds;
EXPLAIN WITH customer_total_return AS
( SELECT sr_customer_sk as ctr_customer_sk,
  sr_store_sk as ctr_store_sk,
  sum(SR_FEE) as ctr_total_return
FROM store_returns, date_dim
WHERE sr_returned_date_sk = d_date_sk AND d_year =2000
GROUP BY sr_customer_sk, sr_store_sk )
SELECT  c_customer_id
FROM customer_total_return ctr1, store, customer
WHERE ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
FROM customer_total_return ctr2
WHERE ctr1.ctr_store_sk = ctr2.ctr_store_sk)
AND s_store_sk = ctr1.ctr_store_sk
AND s_state = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

In the following query result, the value in the first row of the **E-costs** column is the estimated cost of the SQL statement.

Figure 15-1 EXPLAIN result

id	operation	E-rows	E-width	E-costs
1	-> Row Adapter	6	20	153.06
2	-> Vector Limit	6	20	153.06
3	-> Vector Streaming (type: GATHER)	6	20	153.06
4	-> Vector Limit	6	20	152.84
5	-> Vector Sort	6	20	152.84
6	-> Vector Hash Join (7,26)	6	20	152.83
7	-> Vector Streaming(type: REDISTRIBUTE)	6	4	134.57
8	-> Vector Hash Join (9,18)	6	4	134.46
9	-> Vector Hash Join (10,11)	1	44	97.33
10	-> CStore Scan on store	1	4	60.23
11	-> Vector Subquery Scan on ctr1	6	40	37.07
12	-> Vector Hash Aggregate	6	54	37.06
13	-> Vector Streaming(type: REDISTRIBUTE)	6	22	37.04
14	-> Vector Hash Join (15,17)	6	22	37.00
15	-> Vector Streaming(type: BROADCAST)	6	4	18.74
16	-> CStore Scan on date_dim	1	4	18.06
17	-> CStore Scan on store_returns	60	26	18.02
18	-> Vector Hash Aggregate	6	68	37.09
19	-> Vector Subquery Scan on ctr2	6	36	37.07
20	-> Vector Hash Aggregate	6	54	37.06
21	-> Vector Streaming(type: REDISTRIBUTE)	6	22	37.04
22	-> Vector Hash Join (23,25)	6	22	37.00
23	-> Vector Streaming(type: BROADCAST)	6	4	18.74
24	-> CStore Scan on date_dim	1	4	18.06
25	-> CStore Scan on store_returns	60	26	18.02
26	-> CStore Scan on customer	60	24	18.02

In this example, to demonstrate the resource monitoring function of top SQLs, set the value of **resource_track_cost** to **100**, which should be lower than the estimated cost in the **EXPLAIN** query result. For more details, see [resource_track_cost](#).

NOTE

After completing this example, you still need to reset [resource_track_cost](#) to its default value **100000** or a proper value. An overly small parameter value will compromise the database performance.

Step 3 Run SQL statements.

```
SET CURRENT_SCHEMA = tpcds;
WITH customer_total_return AS
(SELECT sr_customer_sk as ctr_customer_sk,
sr_store_sk as ctr_store_sk,
sum(SR_FEE) as ctr_total_return
FROM store_returns,date_dim
WHERE sr_returned_date_sk = d_date_sk
AND d_year =2000
GROUP BY sr_customer_sk ,sr_store_sk)
SELECT c_customer_id
FROM customer_total_return ctr1, store, customer
WHERE ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
FROM customer_total_return ctr2
WHERE ctr1.ctr_store_sk = ctr2.ctr_store_sk)
AND s_store_sk = ctr1.ctr_store_sk
AND s_state = 'TN'
AND ctr1.ctr_customer_sk = c_customer_sk
ORDER BY c_customer_id
limit 100;
```

Step 4 During statement execution, query for the real-time memory peak information about the SQL statement on the current CN.

```
SELECT query,max_peak_memory,average_peak_memory,memory_skew_percent FROM
gs_wlm_session_statistics ORDER BY start_time DESC;
```

The preceding command queries for the real-time peak information at the query-level. The peak information includes the maximum memory peak among all DNs per second, average memory peak among all DNs per second, and memory usage skew across DNs.

For more examples of querying for the real-time resource monitoring information of top SQLs, see [Real-time Top SQL](#).

Step 5 Wait until the SQL statement execution in [Step 3](#) is complete, and then query for the historical resource monitoring information of the statement.

```
SELECT query,start_time,finish_time,duration,status FROM gs_wlm_session_history ORDER BY start_time
desc;
```

The preceding command queries for the historical information at the query-level. The peak information includes the execution start time, end time, actual execution time, and execution status. The time unit is ms.

For more examples of querying for the historical resource monitoring information of top SQLs, see [Historical Top SQL](#).

Step 6 Wait for 3 minutes after the execution of the SQL statement in [Step 3](#) is complete, query for the historical resource monitoring information of the statement in the **info** view.

If **enable_resource_record** is set to **on** and the execution time of the SQL statement in [Step 3](#) is no less than the value of **resource_track_duration**, historical information about the SQL statement will be archived to the **gs_wlm_session_info** view 3 minutes after the execution of the SQL statement is complete.

The **info** view can be queried only when the **postgres** database is connected. Therefore, switch to the **postgres** database before running the following statement:

```
SELECT query,start_time,finish_time,duration,status FROM gs_wlm_session_info ORDER BY start_time desc;
```

----**End**

16 DWS Performance Tuning

16.1 Overview

Database performance tuning is the process of optimizing database system configuration and SQL queries to improve database performance and efficiency. The purpose includes eliminating performance bottlenecks, reducing response times, increasing throughput and resource utilization, cutting costs, and improving system stability.

This section provides comprehensive guidance for DBAs on performance diagnosis, system tuning, and SQL tuning, as well as practical examples of SQL tuning.

Precautions

- Database performance tuning is a complex and intricate process. To achieve the optimal performance and efficiency, performance tuning must take into consideration multiple factors, such as hardware, software, queries, configuration, and data structures. Engineers performing the performance tuning must be familiar with how database systems work in great detail, including a deep understanding of the system software architecture, software and hardware configurations, database configuration parameters, concurrency control, query handling, and database applications.
- Performance tuning sometimes requires a cluster restart, which may interrupt services. To avoid that, you are advised to schedule performance tuning tasks that require a cluster restart to occur during off-peak hours.

Performance Tuning Process

[Figure 16-1](#) illustrates the performance tuning process.

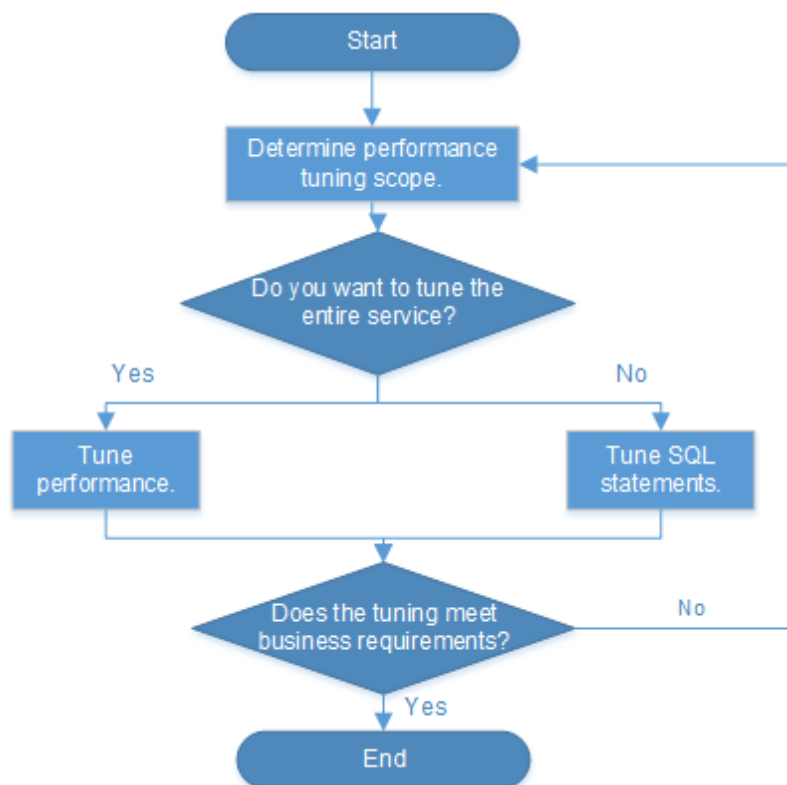
Figure 16-1 DWS performance tuning

Table 16-1 gives a brief introduction to each phase of the performance tuning process.

Table 16-1 Phase-by-phase introduction to DWS performance tuning

Phase	Description
Performance Diagnosis	Obtain the CPU, memory, I/O, and network resource usage of each node to check whether these resources are fully utilized and whether any performance bottlenecks exist.
System Optimization	Perform OS and database system-level performance tuning to achieve better utilization of existing CPU, memory, I/O, and network resources, prevent resource conflicts, and improve query throughput.

Phase	Description
SQL Tuning	Analyze the SQL statements used and determine whether any optimization can be performed. Analysis of SQL statements comprises: • Generating table statistics using ANALYZE: The ANALYZE statement collects statistics about the database table content. Statistical results are stored in the system catalog PG_STATISTIC. The execution plan generator uses these statistics to determine which one is the most effective execution plan. • Analyzing the execution plan: The EXPLAIN statement displays the execution plan of SQL statements, and the EXPLAIN PERFORMANCE statement displays the execution time of each operator in SQL statements. • Identifying the root causes of issues: Identify possible causes by analyzing the execution plan and perform specific optimization by modifying database-level SQL optimization parameters. • Compiling better SQL statements: Compile better SQL statements in the scenarios, such as cache of intermediate and temporary data for complex queries, result set cache, and result set combination.

16.2 Performance Diagnosis

16.2.1 Cluster Performance Analysis

The node specifications of different DWS clusters may vary in terms of the number of CPU cores, memory capacity, and node storage capacity. Different specifications lead to different service handling capacity and performance. Before creating a cluster, you need to select the appropriate cluster specifications based on the actual workloads and application scenario.

If the workloads increase, more resources (such as CPU, memory, and network bandwidth) will be needed in order to maintain the same level of database performance. Insufficient cluster resources will lead to performance issues.

DWS provides abundant monitoring metrics that you can use to monitor cluster performance and status, including CPU usage, memory usage, disk usage, disk I/O, and network I/O. For any abnormality, you can check the metrics to locate the root cause.

If your service requires additional compute or storage resources, expand the capacity of an existing cluster by adding more nodes to it or changing node specifications through the management console.

16.2.2 Slow SQL Analysis

16.2.2.1 Querying SQL Statements That Affect Performance Most

This section describes how to query SQL statements whose execution takes a long time, leading to poor system performance.

Procedure

Step 1 Query the statements that are run for a long time in the database.

```
SELECT current_timestamp - query_start AS runtime, datname, username, query FROM pg_stat_activity  
where state != 'idle' ORDER BY 1 desc;
```

After the query, query statements are returned as a list, ranked by execution time in descending order. The first result is the query statement that has the longest execution time in the system. The returned result contains the SQL statement invoked by the system and the SQL statement run by users. Find the statements that were run by users and took a long time.

Alternatively, you can set **current_timestamp - query_start** to be greater than a threshold to identify query statements that are executed for a duration longer than this threshold.

```
SELECT query FROM pg_stat_activity WHERE current_timestamp - query_start > interval '1 days';
```

Step 2 Set the parameter **track_activities** to **on**.

```
SET track_activities = on;
```

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

Step 3 View the running query statements.

Viewing **pg_stat_activity** is used as an example here.

```
SELECT datname, username, state FROM pg_stat_activity;  
datname | username | state |  
-----+-----+-----+  
postgres | omm      | idle  |  
postgres | omm      | active|  
(2 rows)
```

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

To identify only active query statements, run the following command:

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

Step 4 Analyze the status of the query statements that were run for a long time.

- If the query statement is normal, wait until the execution is complete.
- If a query statement is blocked, run the following command to view this query statement:

```
SELECT datname, username, state, query FROM pg_stat_activity WHERE waiting = true;
```

The command output lists a query statement in the block state. The lock resource requested by this query statement is occupied by another session, so this query statement is waiting for the session to release the lock resource.

 NOTE

Only when the query is blocked by internal lock resources, the **waiting** field is **true**. In most cases, block happens when query statements are waiting for lock resources to be released. However, query statements may be blocked because they are waiting to write in files or for timers. Such blocked queries are not displayed in the **pg_stat_activity** view.

----End

16.2.2.2 Checking Blocked Statements

During database running, query statements are blocked in some service scenarios and run for an excessively long time. In this case, you can forcibly terminate the faulty session.

Procedure

- Step 1** View blocked query statements and information about the tables and schemas that block the query statements.

```
SELECT w.query as waiting_query,  
       w.pid as w_pid,  
       w.username as w_user,  
       l.query as locking_query,  
       l.pid as l_pid,  
       l.username as l_user,  
       t.schemaname || '.' || t.relname as tablename  
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid  
and not l1.granted join pg_locks l2 on l1.relation = l2.relation  
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t on l1.relation = t.relid  
where w.waiting;
```

The thread ID, user information, query status, as well as information about the tables and schemas that block the query statements are returned.

- Step 2** Run the following command to terminate the required session, where **139834762094352** is the thread ID:

```
SELECT PG_TERMINATE_BACKEND(139834762094352);
```

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

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

If a command output similar to the following is displayed, a user is attempting to terminate the session, and the session will be reconnected rather than being terminated.

```
FATAL: terminating connection due to administrator command  
FATAL: terminating connection due to administrator command  
The connection to the server was lost. Attempting reset: Succeeded.
```

 NOTE

If the **PG_TERMINATE_BACKEND** function is used to terminate the background threads of the session, the **gsql** client will be reconnected rather than be logged out.

----End

16.2.3 SQL Diagnosis

DWS clusters support SQL diagnosis, which shows the complete execution plans of specific SQL queries. You can search for SQL queries (such as slow queries) using a combination of multiple filter criteria.

To use SQL diagnosis, perform the following steps:

- Step 1** Log in to the DWS console.
- Step 2** Locate the cluster to be monitored.
- Step 3** In the **Operation** column of the target cluster, click **Monitoring Panel**.
- Step 4** In the navigation pane on the left, choose **Utilities > SQL Diagnosis**.
- Step 5** On the **SQL Diagnosis** page, you can view the SQL diagnosis information. For details about the metrics, see [Table 16-2](#).

Table 16-2 SQL diagnosis metrics

Metric	Description
Query ID	Internal query_ID used for statement execution
Databases	Database name
Schema Name	Name of the schema
Username	Username
Client	Name of the application connected to the backend
Client IP Address	IP address of the client connected to the backend. If this column is null, it indicates that the client is connected via a Unix socket on the server machine or that it is an internal process, such as autovacuum .
Running Time (ms)	Execution time of the statement, in milliseconds
CPU Time (ms)	Total CPU time of a statement across all DNs, in milliseconds
Start Time	Time when the statement starts to be executed
Completed	End time of statement execution
Details	Click View to view the detailed diagnosis result of the SQL statement. including: - Alarm information about the SQL statement - SQL statement - Execution plan of the SQL statement

----End

16.2.4 Table Diagnosis

DWS provides statistics and diagnostic tools for you to learn table status, including:

- **Skew Rate:** monitors and analyzes uneven data distribution in a cluster, and displays information about the 50 largest tables whose skew rate is higher than 5%.
- **Dirty Page Rate:** monitors and analyzes dirty pages in a cluster, and displays information about the 50 largest tables whose dirty page rate is higher than 50%.

Skew Rate

Improper distribution columns can cause severe skew during operator computing or data spill to disk. The workloads will be unevenly distributed on DN, resulting in high disk usage on individual DN and affecting their performance. After identifying tables with a high skew rate and a relatively large size, you can reselect distribution columns for these tables to have their data redistributed.

Procedure

- Step 1** Log in to the DWS console.
- Step 2** Locate the cluster to be monitored.
- Step 3** In the **Operation** column of the target cluster, click **Monitoring Panel**.
- Step 4** In the navigation pane on the left, choose **Tools > Table Diagnosis** and click the **Skew Rate** tab. The tables that meet the statistics collection conditions in the cluster are displayed (top 50 tables with the skew rate higher than 5%).

This page uses a unique table skew rate calculation method. It computes the skew rate using this formula: (Largest table size per DN – Smallest table size per DN)/Average table size per DN.

To find the needed values, check the **dnsize** column in the **gs_table_distribution** view.

- Step 5** (Optional) On the **Skew Rate** page, click **Export** to export the table information that meets the statistical conditions to a CSV file. You can open the CSV file in a local path.

----End

Dirty Page Rate

DML operations on tables may generate dirty data, which unnecessarily occupies cluster storage. You can identify tables with a high dirty page rate and a relatively large size, and handle them accordingly.

Procedure

- Step 1** Log in to the DWS console.
- Step 2** Locate the cluster to be monitored.

- Step 3** In the **Operation** column of the target cluster, click **Monitoring Panel**.
- Step 4** In the navigation pane on the left, choose **Tools > Table Diagnosis** and click the **Dirty Page Rate** tab. The tables that meet the statistics collection conditions in the cluster are displayed.
- Step 5** (Optional) On the **Dirty Page Rate** page, click **Export** to export the table information that meets the statistical conditions to a CSV file. You can open the CSV file in a local path.

----End

16.3 System Optimization

16.3.1 Tuning Database Parameters

To ensure high performance of the database, you are advised to configure GUC parameters based on available resources and the actual workloads. This section describes some of the common parameters and the recommended configurations for them. For more details, see [Configuring GUC Parameters](#).

Parameters Related to Database Memory

Table 16-3 Parameters related to database memory

GUC Parameter	Description	Suggestion
max_process_memory	Specifies the maximum physical memory available to a single CN/DN.	- On DNs, the value of this parameter is determined based on the server's physical memory and the number of DNs deployed on a single node. Parameter value = (Physical memory - vm.min_free_kbytes) x 0.8/(n + Number of primary DNs). This parameter aims to ensure system reliability, preventing node OOM caused by increasing memory usage. vm.min_free_kbytes indicates OS memory reserved for kernels to receive and send data. Its value is at least 5% of the total memory. That is, max_process_memory = Physical memory x 0.8/ (n + Number of primary DNs). If the cluster scale (number of nodes in the cluster) is smaller than 256, n=1; if the cluster scale is larger than 256 and smaller than 512, n=2; if the cluster scale is larger than 512, n=3. - Set this parameter on CNs to the same value as that on DNs. - RAM is the maximum memory allocated to the cluster.

GUC Parameter	Description	Suggestion
shared_buffers	Specifies the size of the shared memory used by DWS. If the value of this parameter is increased, DWS requires more System V shared memory than the default system setting.	It is recommended that shared_buffers be set to a value less than 40% of the memory. Set it to a large value for row-store tables and a small value for column-store tables. Set this parameter to a large value for row storage and a small value for column storage. For column-store tables: $\text{shared_buffers} = (\text{Memory of a single server} / \text{Number of DNs on the single server}) \times 0.4 \times 0.25$ If you want to increase the value of shared_buffers, you also need to increase the value of checkpoint_segments, because a longer period of time is required to write a large amount of new or changed data.
cstore_buffers	Specifies the size of the shared buffer used by column-store tables and column-store tables (ORC, Parquet, and CarbonData) of OBS and HDFS foreign tables.	Column-store tables use the shared buffer specified by cstore_buffers instead of that specified by shared_buffers. When column-store tables are mainly used, reduce the value of shared_buffers and increase that of cstore_buffers. Use cstore_buffers to specify the cache of ORC, Parquet, or CarbonData metadata and data for OBS or HDFS foreign tables. The metadata cache size should be 1/4 of cstore_buffers and not exceed 2 GB. The remaining cache is shared by column-store data and foreign table column-store data.

GUC Parameter	Description	Suggestion
work_mem	Specifies the size of the memory used by internal sequential operations and the Hash table before data is written into temporary disk files.	The default value is 512 MB for small-scale memory (max_process_memory is less than 30 GB) and 2 GB for large-scale memory (max_process_memory is greater than or equal to 30 GB). When the specified physical memory is insufficient, work_mem determines whether to write additional operator calculation data into temporary tables based on query characteristics and concurrency. This reduces performance by five to ten times and increases query response times from seconds to minutes. • In complex serial query scenarios, each query requires five to ten associated operations. Set work_mem using the following formula: work_mem = 50% of the memory/10. • In simple serial query scenarios, each query requires two to five associated operations. Set work_mem using the following formula: work_mem = 50% of the memory/5. • For concurrent queries, use the formula: work_mem = work_mem in serialized scenario/Number of concurrent SQL statements.

GUC Parameter	Description	Suggestion
maintenance_work_mem	Specifies the maximum size of memory used for maintenance operations, involving VACUUM , CREATE INDEX , and ALTER TABLE ADD FOREIGN KEY .	If you set this parameter to the value of work_mem, database dump files can be cleaned up and restored more efficiently. In a database session, only one maintenance operation can be performed at a time. Maintenance is usually performed when there are not much sessions. When the automatic cleanup process is running, up to autovacuum_max_workers times of the memory will be allocated. In this case, set maintenance_work_mem to a value greater than or equal to that of work_mem.

Parameters Related to Queue Concurrency in Databases

GUC Parameter	Description	Suggestion
max_active_statements (global concurrent queue)	Controls the maximum number of concurrent jobs on a single CN.	All common users' jobs are subject to this threshold, regardless of their complexity. When the number of concurrent jobs reaches the specified threshold, the excess jobs have to wait in a queue. Administrator's jobs are exempt from this limit. Set the value of this parameter based on system resources, such as CPU, I/O, and memory resources, to ensure that the system resources can be fully utilized and the system will not be crashed due to excessive concurrent jobs.
parctl_min_cost (local concurrent queue)	Controls the maximum number of concurrent jobs within the same resource pool on a single CN.	The number of concurrent complex jobs are controlled based on their cost.

 NOTE

When tuning the **max_active_statements** parameter (global concurrent queue), pay attention to the following:

- If **max_active_statements** is set to **-1**, which indicates that global concurrency is not limited, users may be disconnected in a high concurrency scenario.
- In a point query scenario, set **max_active_statements** to **100**.
- In an analytical query scenario, set **max_active_statements** to the number of CPU cores divided by the number of DNs. Generally, its value ranges from 5 to 8.

Database Communication Parameters

By default, nodes in a database cluster communicate using the TCP proxy communication library.

Table 16-4 Database communication parameters

GUC Parameter	Description	Suggestion
comm_quota_size	comm_quota_size controls the size of data transmitted every time in each flow channel. Its default value is 1M .	In a high concurrency scenario, you can increase its value to improve communication performance, but doing so consumes more memory. Optimize this parameter as needed. If you query the pg_total_memory_detail view of a DN and find that the memory used by the communication layer has reached the threshold of comm_usable_memory , set comm_quota_size to a small value, such as 512K .
comm_usable_memory	comm_usable_memory controls the memory on a DN that can be used for database communication.	The value of this parameter is only used for memory flow control. The default flow control value is 1 MB. If the memory usage exceeds half of the parameter value, the flow control value will be automatically changed to 0.5 MB. If only 20% of the memory specified by the parameter is available, the flow control value will be changed to the allowed minimum, 8 KB.

Database Connection Parameters

Table 16-5 Database connection parameters

GUC Parameter	Description	Suggestion
max_connections	Specifies the maximum number of concurrent connections to the database. This parameter affects the concurrent processing capability of the cluster.	Retain the default value of this parameter on CNs. Set this parameter on DN to a value calculated using this formula: Number of CNs x Value of this parameter on a CN. If the value of this parameter is increased, DWS may require more System V shared memory or semaphore, which may exceed the default maximum value of the OS. In this case, modify the value as needed.
max_prepared_transactions	Specifies the maximum number of transactions that can stay in the prepared state simultaneously. If the value of this parameter is increased, DWS requires more System V shared memory than the default system setting.	The value of max_connections is related to max_prepared_transactions . Before configuring max_connections , ensure that the value of max_prepared_transactions is greater than or equal to that of max_connections . In this way, each session has a prepared transaction in the waiting state.
session_timeout	Specifies the maximum duration a database connection can stay idle before it is automatically disconnected.	The value can be an integer in the range 0 to 86400. The minimum unit is second (s). The value 0 disables this timeout mechanism. Generally, you are advised not to set this parameter to 0 .

Other Performance-related Parameters

Table 16-6 Other performance-related parameters

GUC Parameter	Description	Suggestion
enable_dynamic_workload	Specifies whether to enable dynamic load management. Dynamic load management refers to the automatic queue control of complex queries based on user loads in a database. This fine-tunes system parameters without manual adjustment.	This parameter is enabled by default. Notes: • Simple query jobs (which are estimated to require less than 32 MB memory) and non-DML statements (statements other than INSERT, UPDATE, DELETE, and SELECT) have no adaptive load restrictions. Control the upper memory limits for them on a single CN using max_active_statements. • In adaptive load scenarios, the value cannot be increased. If you increase it, memory cannot be controlled for certain statements, such as statements that have not been analyzed. • Reduce concurrency in the following scenarios, because high concurrency may lead to uncontrollable memory usage. – A single tuple occupies excessive memory, for example, a base table contains a column more than 1 MB wide. – A query is fully pushed down. – A statement occupies a large amount of memory on the CN, for example, a statement that cannot be pushed down or a cursor withholding statement. – An execution plan creates a hash table based on the hash join operator, and the table has many duplicate values and occupies a large amount of memory. – UDFs are used, which occupy a large amount of memory. • When configuring this parameter, you can set query_dop to 0 (adaptive). In this case, the system dynamically selects the optimal degree of parallelism (DOP) for

GUC Parameter	Description	Suggestion
		each query based on resource usage and the execution plan. The enable_dynamic_workload parameter supports the dynamic memory allocation.
bulk_write_ring_size	Specifies the size of a ring buffer used for parallel data import.	This parameter affects the database import performance. You are advised to increase the value of this parameter on DN nodes when a large amount of data is to be imported. The default value is 2GB .
data_replicate_buffer_size	Specifies the memory used by queues when the sender sends data pages to the receiver.	The value of this parameter affects the buffer size for data replication between the primary and standby servers. The default value is 16 MB for a CN and 128 MB for a DN. If the server memory is 256 GB, you can increase the value to 512 MB.

16.3.2 SMP Parallel Execution

Complex queries may take a long time. In a system with low concurrency support, this can be a problem. SMP is used to implement operator-level parallel execution, which can effectively speed up queries, improving query performance and resource utilization.

The SMP feature improves performance through operator parallelism but may drive more resource usage, including CPU, memory, network, and I/O. In essence, SMP is a method that trades resources for time, meaning it accelerates queries at the cost of additional resources. It improves system performance in appropriate scenarios and when resources are sufficient, but may also deteriorate performance if used inappropriately. Furthermore, compared with serial processing, SMP generates more candidate plans, which is more time-consuming and may hurt performance.

The SMP feature of DWS is controlled by the GUC parameter **query_dop**. Users use this parameter to specify an appropriate degree of query parallelism.

Application Scenarios and Constraints for SMP

Applicable Scenarios

- Operators supporting parallel processing are used.

The execution plan contains the following operators:

- Scan: Row Storage common table and a line memory partition table sequential scanning, column-oriented storage ordinary table and column-

oriented storage partition table sequential scanning, HDFS internal and external table sequence scanning. Surface scanning GDS data can be imported at the same time. All of the above does not support replication tables.

- b. Join: HashJoin, NestLoop
- c. Agg: HashAgg, SortAgg, PlainAgg, and WindowAgg, which supports only **partition by**, and does not support **order by**.
- d. Stream: Redistribute, Broadcast
- e. Other: Result, Subqueryscan, Unique, Material, Setop, Append, VectorRow, RowToVec
- SMP-unique operators are used.
To execute queries in parallel, Stream operators are added for data exchange for the SMP feature. These new operators can be considered as the subtypes of Stream operators.
 - a. Local Gather aggregates data of parallel threads within a DN
 - b. Local Redistribute redistributes data based on the distributed key across threads within a DN
 - c. Local Broadcast broadcasts data to each thread within a DN.
 - d. Local RoundRobin distributes data in polling mode across threads within a DN.
 - e. Split Redistribute redistributes data across parallel threads on different DNs.
 - f. Split Broadcast broadcasts data to all parallel DN threads in the cluster.

Among these operators, Local operators exchange data between parallel threads within a DN, and non-Local operators exchange data across DNs.

- Example

The TPCB Q1 parallel plan is used as an example.

```

id | operation
-----+-----
1 | -> Row Adapter
2 |   -> Vector Streaming (type: GATHER)
3 |     -> Vector Sort
4 |       -> Vector Streaming(type: LOCAL GATHER dop: 1/4)
5 |         -> Vector Hash Aggregate
6 |           -> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4)
7 |             -> Vector Hash Aggregate
8 |               -> Vector Append(9, 10)
9 |                 -> Dfs Scan on lineitem
10 |                  -> Vector Adapter
11 |                    -> Seq Scan on pg_delta_1423863972 lineitem
(11 rows)

```

In this plan, implement the Hdfs Scan and HashAgg operator parallel, and adds the Local Gather and Split Redistribute data exchange operator.

In this example, the sixth operator is Split Redistribute, and **dop: 4/4** next to the operator indicates that the degree of parallelism of the sender and receiver is 4. 4 No operator is Local Gather, marked dop: 1/4 above, this operator sender thread parallel degree is 4, while the receiving end thread parallelism degree to 1, that is, lower-layer 5 number Hash Aggregate operators according to the 4 parallel degree, while the working mode of the port on the upper-layer 1 to 3 number operator according to the executed

one by one, 4 number operator is used to achieve intra-DN concurrent threads data aggregation.

You can view the parallelism situation of each operator in the dop information.

Non-Applicable Scenarios

1. Small queries are performed, where plan generation may account for a significant portion of the total query time.
2. Operators are processed on CNs.
3. Statements that cannot be pushed down are executed.
4. The **subplan** of a query and operators containing a subquery are executed.

Impact of Resource Availability on SMP Performance

The SMP architecture accelerates queries at the cost of additional resources. After the plan parallelism is executed, more resources are consumed, including the CPU, memory, I/O, and network bandwidth. As the DOP grows, the resource consumption also increases. If these resources become a bottleneck, SMP cannot improve performance. On the contrary, it may do exactly the opposite. Adaptive SMP is provided to dynamically select the optimal parallel degree for each query based on the resource usage and query requirements. The following information describes the situations that the SMP affects these resources:

- **CPU resources**

In a general customer scenario, the system CPU usage rate is not high. Using the SMP parallelism architecture will fully use the CPU resource to improve the system performance. If the number of CPU kernels of the database server is too small and the CPU usage is already high, enabling the SMP parallelism may deteriorate the system performance due to resource compete between multiple threads.

- **Memory resources**

The query parallel causes memory usage growth, but the memory upper limit used by each operator is still restricted by **work_mem**. Assume that **work_mem** is 4 GB, and the degree of parallelism is 2, then the memory upper limit of each concurrent thread is 2 GB. When **work_mem** is small or the system memory is sufficient, running SMP parallelism may push data down to disks. As a result, the query performance deteriorates.

- **Network bandwidth resources**

To execute queries in parallel, data exchange operators are added. Local stream operators exchange data between threads within a DN. Data is exchanged in memory, so it does not impact network performance. Non-local operators exchange data over the network and increase the network load. If the capacity of a network resource has already become a bottleneck, parallelism may hurt performance.

- **I/O resources**

A parallel scan increases I/O resource consumption. It can improve performance only when I/O resources are sufficient.

Other Factors Impacting SMP Performance

Besides the resource factor, other factors may also impact SMP performance, such as uneven data distribution across tables and the degree of system parallelism.

- **Impact of data skew on SMP performance**

Serious data skew deteriorates parallel execution performance. For example, if the data volume of a value in the join column is much more than that of other values, the data volume of a parallel thread will be much more than that of others after Hash-based data redistribution, resulting in the long-tail issue and poor parallelism performance.

- **Impact of system parallelism degree on SMP performance**

The SMP feature uses more resources, and unused resources are decreasing in a high concurrency scenario. Therefore, enabling the SMP parallelism will result in serious resource compete among queries. Once resource competes occur, no matter the CPU, I/O, memory, or network resources, all of them will result in entire performance deterioration. In the high concurrency scenario, enabling the SMP will not improve the performance effect and even may cause performance deterioration.

Suggestions for SMP Parameter Settings

To enable the SMP adaptation function, set **query_dop** to **0** and adjust the following parameters to obtain an optimal DOP selection:

- **comm_usable_memory**

If the system memory is large, the value of **max_process_memory** is large. In this case, you are advised to set the value of this parameter to 5% of **max_process_memory**, that is, 4 GB by default.

- **comm_max_stream**

The recommended value for this parameter is calculated as follows:
 $\text{comm_max_stream} = \text{Min}(\text{dop_limit} \times \text{dop_limit} \times 20 \times 2, \text{max_process_memory (bytes)} \times 0.025 / \text{Number of DNs} / 260)$. The value must be within the value range of **comm_max_stream**.

- **max_connections**

The recommended value for this parameter is calculated as follows:
 $\text{max_connections} = \text{dop_limit} \times 20 \times 6 + 24$. The value must be within the value range of **max_connections**.

CAUTION

In the preceding formulas, **dop_limit** indicates the number of CPUs corresponding to each DN in the cluster. It is calculated as follows: **dop_limit** = Number of logical CPU cores of a single server / Number of DNs of a single server.

SMP Configuration Procedure

NOTICE

The CPU, memory, I/O, and network bandwidth resources are sufficient. In essence, SMP is a method that trades resources for time. After the plan parallelism is executed, resource consumption increases. When these resources become a bottleneck, SMP may deteriorate, rather than improve performance. In addition, it takes a longer time to generate SMP plans than serial plans. Therefore, in TP services that mainly involve short queries or in case resources are insufficient, you are advised to disable SMP by setting **query_dop** to 1.

Procedure:

1. Observe the current system load situation. If the resource is sufficient (the resource usage ratio is smaller than 50%), perform step 2. Otherwise, exit this system.
2. Set **query_dop** to 1 (default value). Use **explain** to generate an execution plan and check whether the plan can be used in scenarios described in [Application Scenarios and Constraints for SMP](#). If the plan can be used, go to the next step.
3. Set **query_dop**=*value*. The value range of the parallelism degree is [1, *value*].
4. Set **query_dop**=*value*. The parallelism degree is 1 or *value*.
5. Before the query statement is executed, set **query_dop** to an appropriate value. After the statement is executed, set **query_dop** to **off**. For example:

```
SET query_dop = 0;  
SELECT COUNT(*) FROM t1 GROUP BY a;  
.....  
SET query_dop = 1;
```

NOTE

- If resources are sufficient, the higher the degree of parallelism, the better the performance improvement result.
- The SMP parallelism degree supports a session level setting and you are advised to enable SMP before executing queries that meet the requirements. After the execution is complete, disable SMP. Otherwise, SMP may affect services during peak hours.
- SMP adaptability (**query_dop** ≤ 0) depends on resource management. If resource management is disabled, only plans with parallelism degree of only 1 or 2 will be generated.

16.3.3 Configuring LLVM

LLVM dynamic compilation can be used to generate customized machine code for each query to replace original common functions. The query performance is improved by reducing redundant judgment condition and virtual function invocation, and make local data more accurate during actual queries.

LLVM needs to consume extra time to pre-generate intermediate representation (IR) and compile it into code. Therefore, if the data volume is small or if a query itself consumes little time, LLVM actually does more harm than good.

LLVM Application Scenarios and Constraints

Applicable Scenarios

- Expressions supporting LLVM. The query statements that contain the following expressions support LLVM optimization:
 - a. CASE...WHEN...
 - b. IN
 - c. Bool (AND/OR/NOT)
 - d. BooleanTest (IS_NOT_UNKNOWN/IS_UNKNOWN/IS_TRUE/IS_NOT_TRUE/IS_FALSE/IS_NOT_FALSE)
 - e. NullTest (IS_NOT_NULL/IS_NULL)
 - f. Operators
 - g. Functions (lpad, substring, btrim, rtrim, and length)
 - h. Nullif

The following data types are supported for expression calculation: bool, tinyint, smallint, int, bigint, float4, float8, numeric, date, time, timetz, timestamp, timestampz, interval, bpchar, varchar, text, and oid.

Consider using LLVM dynamic compilation and optimization only when expressions are used in the following scenarios:

- **filter** on the **Scan** node in the case of a vectorized executor.
- **complicate hash condition**, **hash join filter**, and **hash join target** in the **Hash Join** node.
- **filter** and **join filter** in the **Nested Loop** node.
- **merge join filter** and **merge join target** in the **Merge Join** node.
- **filter** in the Group node.
- Operators that can use LLVM:
 - a. Join: HashJoin
 - b. Agg: HashAgg
 - c. Sort

Among them:

- HashJoin supports only Hash Inner Join, and the corresponding hash cond supports comparisons between int4, bigint, and bpchar.
- HashAgg supports sum and avg operations of bigint and numeric data types. Group By statements support int4, bigint, bpchar, text, varchar, timestamp, and the count(*) aggregation operation.
- Sort supports only comparisons between int4, bigint, numeric, bpchar, text, and varchar data types.

With the exception of the operations above, LLVM dynamic compilation and optimization cannot be used. To further confirm, use the explain performance tool to check.

Non-Applicable Scenarios

- LLVM dynamic compilation and optimization are not supported on CNs.

- Tables that have small amounts of data cannot be dynamically compiled using LLVM.
- Query jobs with a non-vectorized execution path cannot be generated.

Other Factors Impacting LLVM Performance

The result of LLVM optimization depends not only on operations and computation in the database, but also on the hardware environment.

- Number of C- functions invoked by query statements
CodeGen cannot be used in all expressions in an entire expression, that is, some expressions use CodeGen while others invoke original C codes for computation. In an entire expression, if more expressions invoke original C codes, LLVM dynamic compilation and optimization may reduce the computational performance. By setting **log_min_messages** to **DEBUG1**, you can check expressions that directly invoke C codes.
- Memory resources
One of the key LLVM features is to ensure the locality of data, that is, data should be stored in registers whenever possible. Data loading should be reduced at the same time. Therefore, when using LLVM optimization, the value of **work_mem** must be set as large as required to ensure that the code is processed in the memory using LLVM. Otherwise, performance may deteriorate.
- Optimizer cost estimation
The LLVM feature realizes a simple cost estimation model. You can determine whether to use LLVM dynamic compilation and optimization for the current node based on the sizes of tables involved in node computation. If the optimizer understates the actual number of rows involved, the expected performance gains may not be realized. An overestimation will have the same effect.

Recommended Usage of LLVM

LLVM is enabled in the database kernel by default, and users can configure it based on the analysis above. The overall suggestions are as follows:

1. Set an appropriate value for **work_mem** and set it as large as possible. If much data is flushed to disks, you are advised to disable LLVM dynamic compilation and optimization by setting **enable_codegen** to **off**.
2. Set an appropriate value for **codegen_cost_threshold** (The default value is 10,000). Ensure that LLVM dynamic compilation and optimization is not used when the data volume is small. After the value is set, if the database performance deteriorates due to the use of LLVM dynamic compilation and optimization, increase the value.
3. If a large number of C- functions are invoked, you are advised to disable LLVM dynamic compilation and optimization.
4. The constants following the **In** expression cannot exceed 10. Otherwise, LLVM compilation and optimization cannot be performed.

NOTE

If resources are sufficient, the database performance will improve as the data volume increases.

16.4 SQL Tuning

16.4.1 SQL Query Execution Process

The process from receiving SQL statements to the statement execution by the SQL engine is shown in [Figure 16-2](#) and [Table 16-7](#). The texts in red are steps where database administrators can optimize queries.

Figure 16-2 Execution process of query-related SQL statements by the SQL engine

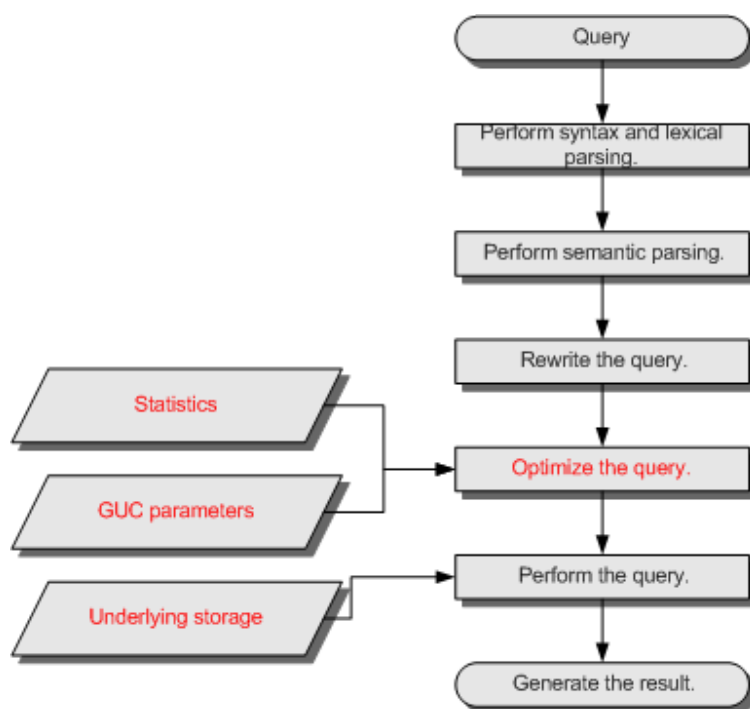


Table 16-7 Execution process of query-related SQL statements by the SQL engine

Step	Description
1. Perform syntax and lexical parsing.	Converts the input SQL statements from the string data type to the formatted structure stmt based on the specified SQL statement rules.
2. Perform semantic parsing.	Converts the formatted structure obtained from the previous step into objects that can be recognized by the database.
3. Rewrite the query statements.	Converts the output of the last step into the structure that optimizes the query execution.

Step	Description
4. Optimize the query.	Determines the execution mode of SQL statements (the execution plan) based on the result obtained from the last step and the internal database statistics. For details about the impact of statistics and GUC parameters on query optimization (execution plan), see Optimizing Queries Using Statistics and Optimizing Queries Using GUC parameters .
5. Perform the query.	Executes the SQL statements based on the execution path specified in the last step. Selecting a proper underlying storage mode improves the query execution efficiency. For details, see Optimizing Queries Using the Underlying Storage .

Optimizing Queries Using Statistics

The DWS optimizer is a typical Cost-based Optimization (CBO). The database uses the CBO to calculate the number of tuples and execution cost for each execution step in every execution plan. This calculation is based on factors such as the number of table tuples, column width, NULL record ratio, and characteristic values (such as distinct, MCV, and HB values) using specific cost calculation methods. The database then selects the execution plan with the lowest cost for overall execution or for returning the first tuple. These characteristic values are the statistics, which is the core for optimizing a query. Accurate statistics helps the optimizer select the most appropriate query plan. Generally, you can collect statistics of a table or that of some columns in a table using **ANALYZE**. You are advised to periodically execute **ANALYZE** or execute it immediately after you modified most contents in a table.

Optimizing Queries Using GUC parameters

Optimizing queries aims to select an efficient execution mode.

Take the following statement as an example:

```
SELECT count(1)
FROM customer inner join store_sales on (ss_customer_sk = c_customer_sk);
```

During execution of **customer inner join store_sales**, DWS supports nested loop, merge join, and hash join. The optimizer estimates the result set value and the execution cost under each join mode based on the statistics of the **customer** and **store_sales** tables and selects the execution plan that takes the lowest execution cost.

As described in the preceding content, the execution cost is calculated based on certain methods and statistics. If the actual execution cost cannot be accurately estimated, you need to optimize the execution plan by setting the GUC parameters.

Optimizing Queries Using the Underlying Storage

DWS supports both row-store and column-store tables. The choice of storage mode ultimately depends on your business needs. Column-store tables are

suitable for computing services that mainly involve associations and aggregations. Row-store tables are better suited for point queries and large-scale updates or deletions.

Optimization methods of each storage mode will be described in details in the performance optimization chapter.

Optimizing Queries by Rewriting SQL Statements

Besides the preceding methods that improve the performance of the execution plan generated by the SQL engine, database administrators can also enhance SQL statement performance by rewriting SQL statements while retaining the original service logic based on the execution mechanism of the database and abundant practical experience.

This requires that the system administrators know the customer business well and have professional knowledge of SQL statements.

16.4.2 SQL Execution Plan

An SQL execution plan is a node tree that displays the detailed steps performed when DWS executes an SQL statement.

You can run the **EXPLAIN** command to view the execution plan generated for each query by an optimizer. **EXPLAIN** outputs a row of information for each execution node, showing the basic node type and the expense estimate that the optimizer makes for executing the node.

Execution Plan Information

In addition to setting different display formats for an execution plan, you can use different **EXPLAIN** syntax to display execution plan information in detail. The common usages are as follows. For more usages, see "EXPLAIN Syntax" in *Data Warehouse Service (DWS) SQL Syntax Reference*.

- **EXPLAIN *statement***: only generates an execution plan and does not execute. The *statement* indicates SQL statements.
- **EXPLAIN ANALYZE *statement***: generates and executes an execution plan, and displays the execution summary. Then actual execution time statistics are added to the display, including the total elapsed time expended within each plan node (in milliseconds) and the total number of rows it actually returned.
- **EXPLAIN PERFORMANCE *statement***: generates and executes the execution plan, and displays all execution information.

To measure the run time cost of each node in the execution plan, the current execution of **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** adds profiling overhead to query execution. Running **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** on a query sometimes takes more time than a normal query. The amount of overhead depends on the nature of the query, as well as the platform being used.

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

EXPLAIN PERFORMANCE statement to check the execution plan and information to locate the fault.

Description of common execution plan keywords:

1. Table access modes

- Seq Scan/CStore Scan

Scans all rows of the table in sequence. These are basic scan operators, which are used to scan row-store and column-store tables in sequence.

- Index Scan/CStore Index Scan

Scans indexes of row-store and column-store tables. There are indexes in row-store or column-store tables, and the condition column is the index column.

The optimizer uses a two-step plan: the child plan node visits an index to find the locations of rows matching the index condition, and then the upper plan node actually fetches those rows from the table itself. Fetching rows separately is much more expensive than reading them sequentially, but because not all pages of the table have to be visited, this is still cheaper than a sequential scan. The upper-layer planning node first sort the location of index identifier rows based on physical locations before reading them. This minimizes the independent capturing overhead.

If there are separate indexes on multiple columns referenced in **WHERE**, the optimizer might choose to use an **AND** or **OR** combination of the indexes. However, this requires the visiting of both indexes, so it is not necessarily a win compared to using just one index and treating the other condition as a filter.

The following Index scans featured with different sorting mechanisms are involved:

- Bitmap Index Scan

To use a bitmap index to capture a data page, you need to scan the index to obtain the bitmap and then scan the base table.

- Index Scan using index_name

Fetches table rows in index order, which makes them even more expensive to read. However, there are so few rows that the extra cost of sorting the row locations is unnecessary. This plan type is used mainly for queries fetching just a single row and queries having an **ORDER BY** condition that matches the index order, because no extra sorting step is needed to satisfy **ORDER BY**.

2. Table connection modes

- Nested Loop

Nested-loop is used for queries that have a smaller data set connected. In a Nested-loop join, the foreign table drives the internal table and each row returned from the foreign table should have a matching row in the internal table. The returned result set of all queries should not exceed 10,000. The table that returns a smaller subset will work as a foreign table, and indexes are recommended for connection fields of the internal table.

- (Sonic) Hash Join

A Hash join is used for large tables. The optimizer uses a hash join, in which rows of one table are entered into an in-memory hash table, after which the other table is scanned and the hash table is probed for matches to each row. Sonic and non-Sonic hash joins differ in their hash table structures, which do not affect the execution result set.

- Merge Join

In a merge join, data in the two joined tables is sorted by join columns. Then, data is extracted from the two tables to a sorted table for matching.

Merge join requires more resources for sorting and its performance is lower than that of hash join. If the source data has been sorted, it does not need to be sorted again when merge join is performed. In this case, the performance of merge join is better than that of hash join.

3. Operators

- sort

Sorts the result set.

- filter

The **EXPLAIN** output shows the **WHERE** clause being applied as a **Filter** condition attached to the **Seq Scan** plan node. This means that the plan node checks the condition for each row it scans, and returns only the ones that meet the condition. The estimated number of output rows has been reduced because of the **WHERE** clause. However, the scan will still have to visit all 10000 rows. As a result, the cost is not decreased. It increases a bit (by 10000 x **cpu_operator_cost**) to reflect the extra CPU time spent on checking the **WHERE** condition.

- LIMIT

LIMIT limits the number of output execution results. If a **LIMIT** condition is added, not all rows are retrieved.

Execution Plan Display Format

DWS provides four display formats for execution plans: normal, pretty, summary, and run. You can change the display format of execution plans by setting **explain_perf_mode**.

- **normal** indicates that the default printing format is used. [Figure 16-3](#) shows the display format.

Figure 16-3 Example of an execution plan in normal format

```
postgres=# explain select * from test where a < 1;
               QUERY PLAN
-----
Streaming (type: GATHER) (cost=0.25..19.16 rows=7 width=8)
  Node/s: All datanodes
    -> Seq Scan on test (cost=0.00..13.16 rows=7 width=8)
        Filter: (a < 1)
(4 rows)
```

- **pretty**: The new updated display format is used. A new format contains a plan node ID, directly and effectively analyzing performance. [Figure 16-4](#) is an example.

Figure 16-4 Example of an execution plan using the pretty format

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

- **summary** indicates that the analysis result based on such information is printed in addition to the printed information in the format specified by **pretty**.
- **run** indicates that in addition to the printed information specified by **summary**, the database exports the information as a CSV file.

Common Types of Plans

DWS has three types of distributed plans:

- **Fast Query Shipping (FQS) plan**
The CN directly delivers statements to DN. Each DN executes the statements independently and summarizes the execution results on the CN.
- **Stream plan**
The CN generates a plan for the statements to be executed and delivers the plan to DNs for execution. During the execution, DNs use the Stream operator to exchange data.
- **Remote-Query plan**
After generating a plan, the CN delivers some statements to DNs. Each DN executes the statements independently and sends the execution result to the CN. The CN executes the remaining statements in the plan.

The existing tables **tt01** and **tt02** are defined as follows:

```
CREATE TABLE tt01(c1 int, c2 int) DISTRIBUTE BY hash(c1);
CREATE TABLE tt02(c1 int, c2 int) DISTRIBUTE BY hash(c2);
```

Type 1: FQS plan, all statements pushed down

Two tables are joined, and the join condition is the distribution column of each table. If the stream operator is disabled, the CN directly sends statements to each DN for execution. The result is summarized on the CN.

```
SET enable_stream_operator=off;
SET explain_perf_mode=normal;

EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c2;
QUERY PLAN
-----
Data Node Scan on "__REMOTE_FQS_QUERY__"
Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
Node/s: All datanodes
Remote query: SELECT tt01.c1, tt01.c2, tt02.c1, tt02.c2 FROM dbadmin.tt01, dbadmin.tt02 WHERE tt01.c1
= tt02.c2
(4 rows)
```

Type 2: Non-FQS plan, some statements pushed down

Two tables are joined and the join condition contains non-distribution columns. If the stream operator is disabled, the CN delivers the base table scanning statements to each DN. Then, the JOIN operation is performed on the CN.

```
SET enable_stream_operator=off;
SET explain_perf_mode=normal;

EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c1;
      QUERY PLAN
-----
Hash Join
  Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
  Hash Cond: (tt01.c1 = tt02.c1)
  -> Data Node Scan on tt01 "_REMOTE_TABLE_QUERY_"
      Output: tt01.c1, tt01.c2
      Node/s: All datanodes
      Remote query: SELECT c1, c2 FROM ONLY dbadmin.tt01 WHERE true
  -> Hash
      Output: tt02.c1, tt02.c2
      -> Data Node Scan on tt02 "_REMOTE_TABLE_QUERY_"
          Output: tt02.c1, tt02.c2
          Node/s: All datanodes
          Remote query: SELECT c1, c2 FROM ONLY dbadmin.tt02 WHERE true
(13 rows)
```

Type 3: Stream plan, no data exchange between DNs

Two tables are joined, and the join condition is the distribution column of each table. DNs do not need to exchange data. After generating a stream plan, the CN delivers the plan except the Gather Stream part to DNs for execution. The CN scans the base table on each DN, performs hash join, and sends the result to the CN.

```
SET enable_fast_query_shipping=off;
SET enable_stream_operator=on;

EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c2;
      QUERY PLAN
-----
Streaming (type: GATHER)
  Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
  Node/s: All datanodes
  -> Hash Join
      Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
      Hash Cond: (tt01.c1 = tt02.c2)
      -> Seq Scan on dbadmin.tt01
          Output: tt01.c1, tt01.c2
          Distribute Key: tt01.c1
      -> Hash
          Output: tt02.c1, tt02.c2
          -> Seq Scan on dbadmin.tt02
              Output: tt02.c1, tt02.c2
              Distribute Key: tt02.c2
(14 rows)
```

Type 4: Stream plan, with data exchange between DNs

When two tables are joined and the join condition contains non-distribution columns, and the stream operator is enabled (SET enable_stream_operator=on), a stream plan is generated, which allows data exchange between DNs. For table **tt02**, the base table is scanned on each DN. After the scanning, the **Redistribute Stream** operator performs hash calculation based on **tt02.c1** in the **JOIN** condition, sends the hash calculation result to each DN, and then performs JOIN on each DN, finally, the data is summarized to the CN.


```

postgres=> SET enable_stream_operator=on;
SET
postgres=> SET enable_fast_query_shipping=off;
SET
postgres=> SET explain_perf_mode=normal;
SET
postgres=> EXPLAIN (VERBOSE on,COSTS off) SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c1;
QUERY PLAN
-----
Streaming (type: GATHER)
  Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
  Node/s: All datanodes
  -> Hash Join
    Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
    Hash Cond: (tt02.c1 = tt01.c1)
    -> Streaming(type: REDISTRIBUTE)
      Output: tt02.c1, tt02.c2
      Distribute Key: tt02.c1
      Spawn on: All datanodes
      Consumer Nodes: All datanodes
      -> Seq Scan on dbadmin.tt02
        Output: tt02.c1, tt02.c2
        Distribute Key: tt02.c2
    -> Hash
      Output: tt01.c1, tt01.c2
      -> Seq Scan on dbadmin.tt01
        Output: tt01.c1, tt01.c2
        Distribute Key: tt01.c1
(19 rows)

```

Type 5: Remote-Query plan

unship_func cannot be pushed down and does not meet partial pushdown requirements (subquery pushdown). Therefore, you can only send base table scanning statements to DNs and collect base table data to the CN for calculation.

```

postgres=> CREATE FUNCTION unship_func(integer,integer) returns integer
postgres-> AS 'select $1 + $2;'
postgres-> LANGUAGE SQL volatile
postgres-> returns null on null input;
CREATE FUNCTION

```

```

postgres=> SET explain_perf_mode=pretty;
SET
postgres=> EXPLAIN VERBOSE SELECT unship_func(tt01.c1,tt01.c2) FROM tt01 JOIN tt02 on tt01.c1=tt02.c1;
QUERY PLAN
-----
id | operation | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+-----
1 | --Hash Join (2,3) | 30 | | 8 | 0.86
2 | --Data Node Scan on tt01 "_REMOTE_TABLE_QUERY_" | 30 | | 8 | 0.00
3 | --Hash | 30 | | 4 | 0.00
4 | --Data Node Scan on tt02 "_REMOTE_TABLE_QUERY_" | 30 | | 4 | 0.00

SQL Diagnostic Information
-----
SQL is not plan-shipping
reason: Function unship_func() can not be shipped

Predicate Information (identified by plan id)
-----
1 --Hash Join (2,3)
Hash Cond: (tt01.c1 = tt02.c1)

Targetlist Information (identified by plan id)
-----
1 --Hash Join (2,3)
Output: (tt01.c1 + tt01.c2)
2 --Data Node Scan on tt01 "_REMOTE_TABLE_QUERY_"
Output: tt01.c1, tt01.c2
Node/s: All datanodes
Remote query: SELECT c1, c2 FROM ONLY dbadmin.tt01 WHERE true
3 --Hash
Output: tt02.c1
4 --Data Node Scan on tt02 "_REMOTE_TABLE_QUERY_"
Output: tt02.c1
Node/s: All datanodes
Remote query: SELECT c1 FROM ONLY dbadmin.tt02 WHERE true

===== Query Summary =====
Parser runtime: 0.055 ms
Planner runtime: 0.528 ms
Unique SQL Id: 1780774145
(37 rows)

```

EXPLAIN PERFORMANCE Description

You can use **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** to check the SQL statement execution information and compare the actual execution and the optimizer's estimation to find what to optimize. **EXPLAIN PERFORMANCE** provides the execution information on each DN, whereas **EXPLAIN ANALYZE** does not.

Tables are defined as follows:

```
CREATE TABLE tt01(c1 int, c2 int) DISTRIBUTE BY hash(c1);  
CREATE TABLE tt02(c1 int, c2 int) DISTRIBUTE BY hash(c2);
```

The following SQL query statement is used as an example:

```
SELECT * FROM tt01,tt02 WHERE tt01.c1=tt02.c2;
```

The output of EXPLAIN PERFORMANCE consists of the following parts:

1. Execution Plan

QUERY PLAN										
id	operation	A-time	A-rows	E-rows	E-distinct	Peak Memory	E-memory	A-width	E-width	E-costs
1	-> Streaming (type: GATHER)	2.566	0	30		24KB			16	36.59
2	-> Hash Join (3,4)	[0.007, 0.009]	0	30	14	[8KB, 8KB]	1MB		16	28.59
3	-> Seq Scan on dbadmin.tt01	[0.002, 0.003]	0	30	14	[16KB, 16KB]	1MB		8	14.14
4	-> Hash	[0, 0]	0	29	14	[0, 0]	16MB		8	14.14
5	-> Seq Scan on dbadmin.tt02	[0, 0]	0	30		[0, 0]	1MB		8	14.14

The plan is displayed as a table, which contains 11 columns: **id**, **operation**, **A-time**, **A-rows**, **E-rows**, **E-distinct**, **Peak Memory**, **E-memory**, **A-width**, **E-width**, and **E-costs**. [Table 16-8](#) describes the columns.

Table 16-8 Execution column description

Column	Description
id	ID of an execution operator.
operation	Name of an execution operator. The operator of the Vector prefix refers to a vectorized execution engine operator, which exists in a query containing a column-store table. Streaming is a special operator. It implements the core data shuffle function of the distributed architecture. Streaming has three types, which correspond to different data shuffle functions in the distributed architecture: • Streaming (type: GATHER): The CN collects data from DNs. • Streaming(type: REDISTRIBUTE): Data is redistributed to all the DNs based on selected columns. • Streaming(type: BROADCAST): Data on the current DN is broadcast to all other DNs.

Column	Description
A-time	Execution time of an operator on each DN. Generally, A-time of an operator is two values enclosed by square brackets ([]), indicating the shortest and longest time for completing the operator on all DNs, including the execution time of the lower-layer operators. Note: In the entire plan, the execution time of a leaf node is the execution time of the operator, while the execution time of other operators includes the execution time of its subnodes.
A-rows	Actual rows output by an operator.
E-rows	Estimated rows output by each operator.
E-distinct	Estimated distinct value of the hash join operator.
Peak Memory	Peak memory used when the operator is executed on each DN. The left value in [] is the minimum value, and the right value in [] is the maximum value.
E-memory	Estimated memory used by each operator on a DN. Only operators executed on DNs are displayed. In certain scenarios, the memory upper limit enclosed in parentheses will be displayed following the estimated memory usage.
A-width	The actual width of each line of tuple of the current operator. This parameter is valid only for the heavy memory operator is displayed, including: (Vec)HashJoin, (Vec)HashAgg, (Vec) HashSetOp, (Vec)Sort, and (Vec)Materialize operator. The (Vec)HashJoin calculation of width is the width of the right subtree operator, it will be displayed in the right subtree.
E-width	Estimated width of the output tuple of each operator.
E-costs	Estimated execution cost of each operator. • E-costs are defined by the optimizer based on cost parameters, habitually grasping disk page as a unit. Other overhead parameters are set by referring to E-costs. • The cost of each node (the E-costs value) includes the cost of all of its child nodes. • Overhead reflects only what the optimizer is concerned about, but does not consider the time that the result row passed to the client. Although the time may play a very important role in the actual total time, it is ignored by the optimizer, because it cannot be changed by modifying the plan.

2. SQL Diagnostic Information

SQL self-diagnosis information. Performance optimization points identified during optimization and execution are displayed. When **EXPLAIN** with the **VERBOSE** attribute (built-in **VERBOSE** of **EXPLAIN PERFORMANCE**) is executed on DML statements, SQL self-diagnosis information is also generated to help locate performance issues.

3. Predicate Information (identified by plan id)

```
Predicate Information (identified by plan id)
-----
2 --Hash Join (3,4)
  Hash Cond: (tt01.c1 = tt02.c2)
3 --Seq Scan on dbadmin.tt01
  Filter: (tt01.c1 >= 202007)
5 --Seq Scan on dbadmin.tt02
  Filter: (tt02.c2 >= 202007)
```

This part displays the filtering conditions of the corresponding execution operator node, that is, the information that does not change during the entire plan execution, mainly the join conditions and filter information.

8.3.0 and later cluster versions support the display the information of **CU Predicate Filter** and **Pushdown Predicate Filter(will be pruned)** related to dictionary plans.

4. Memory Information (identified by plan id)

```
Memory Information (identified by plan id)
-----
Coordinator Query Peak Memory:
  Query Peak Memory: 2MB
DataNode Query Peak Memory
  dn_6001_6002 Query Peak Memory: 0MB
  dn_6003_6004 Query Peak Memory: 0MB
  dn_6005_6006 Query Peak Memory: 0MB
1 --Streaming (type: GATHER)
  Peak Memory: 56KB, Estimate Memory: 512MB
2 --Hash Join (3,4)
  dn_6001_6002 Peak Memory: 8KB, Estimate Memory: 1024KB
  dn_6003_6004 Peak Memory: 8KB, Estimate Memory: 1024KB
  dn_6005_6006 Peak Memory: 8KB, Estimate Memory: 1024KB
3 --Seq Scan on dbadmin.tt01
  dn_6001_6002 Peak Memory: 32KB, Estimate Memory: 1024KB
  dn_6003_6004 Peak Memory: 32KB, Estimate Memory: 1024KB
  dn_6005_6006 Peak Memory: 32KB, Estimate Memory: 1024KB
4 --Hash
  dn_6001_6002 Buckets: 0   Batches: 0   Memory Usage: 0kB
  dn_6003_6004 Buckets: 0   Batches: 0   Memory Usage: 0kB
  dn_6005_6006 Buckets: 0   Batches: 0   Memory Usage: 0kB
```

The memory usage part displays the memory usage information printed by certain operators (mainly Hash and Sort), including **peak memory**, **estimate memory**, **control memory**, **operator memory**, **width**, **auto spread num**, and **early spilled**; and spill details, including **spill Time(s)**, **inner/outer partition spill num**, **temp file num**, spilled data volume, and **written disk I/O [min, max]**. The Sort operator does not display the number of files written to disks, and displays disks only when displaying sorting methods.

5. Targetlist Information (identified by plan id)

```
Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
   Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
   Node/s: All datanodes
2 --Hash Join (3,4)
   Output: tt01.c1, tt01.c2, tt02.c1, tt02.c2
3 --Seq Scan on dbadmin.tt01
   Output: tt01.c1, tt01.c2
   Distribute Key: tt01.c1
4 --Hash
   Output: tt02.c1, tt02.c2
5 --Seq Scan on dbadmin.tt02
   Output: tt02.c1, tt02.c2
   Distribute Key: tt02.c2
```

This part displays the output target column information of each operator.

In 8.3.0 and later cluster versions, the dictionary parameters **Dict Optimized** and **Dict Decoded** can be displayed, indicating dictionary columns and dictionary codes, respectively.

6. DataNode Information (identified by plan id)

```
Datanode Information (identified by plan id)
-----
1 --Row Adapter
   (actual time=68637.768..68637.772 rows=1 loops=1)
   (CPU: ex c/r=7589244, ex row=1, ex c/c=7589244, inc c/c=285913250248)
2 --Vector Aggregate
   (actual time=68635.241..68635.241 rows=1 loops=1) (projection time=0.014)
   (Buffers: shared hit=1)
   (CPU: ex c/r=1675582, ex row=2, ex c/c=3351164, inc c/c=285905661004)
3 --Vector Streaming (type: GATHER)
   (actual time=54126.686..68634.124 rows=2 loops=1)
   (Buffers: shared hit=1)
   (CPU: ex c/r=102951154920, ex row=2, ex c/c=205902309840, inc c/c=205902309840)
4 --Vector Aggregate
   dn_6001_6002 (actual time=54076.307..54076.308 rows=1 loops=1) (projection time=65.581)
   dn_6003_6004 (actual time=68597.121..68597.122 rows=1 loops=1) (projection time=64.801)
   dn_6001_6002 (Buffers: shared hit=1725 dirtied=117)
   dn_6003_6004 (Buffers: shared hit=1723 dirtied=117)
   dn_6001_6002 (CPU: ex c/r=44, ex row=29999729, ex c/c=1346672758, inc c/c=162228870602)
   dn_6003_6004 (CPU: ex c/r=44, ex row=29999728, ex c/c=1343711688, inc c/c=285791299646)
5 --CStore Scan on public.linetime_v3
   dn_6001_6002 (actual time=594.261..53628.924 rows=29999729 loops=1) (CU ScanInfo: smallCu: 0, totalCu: 500, avrCuRow: 59999, totalDeadRows: 0)
   dn_6003_6004 (actual time=865.099..68150.512 rows=29999728 loops=1) (CU ScanInfo: smallCu: 0, totalCu: 500, avrCuRow: 59999, totalDeadRows: 0)
   dn_6001_6002 (Buffers: shared hit=1725 dirtied=117)
   dn_6003_6004 (Buffers: shared hit=1722 dirtied=117)
   dn_6001_6002 (CStore Buffers: shared hit=29500 read=500)
   dn_6003_6004 (CStore Buffers: shared hit=29500 read=500)
   dn_6001_6002 (Disk Cache: miss=576 scanBytes=80MB remoteReadBytes=576MB)
   dn_6003_6004 (Disk Cache: miss=579 scanBytes=80MB remoteReadBytes=579MB)
   dn_6001_6002 (CPU: ex c/r=5362, ex row=29999729, ex c/c=168882197844, inc c/c=168882197844)
   dn_6003_6004 (CPU: ex c/r=6814, ex row=29999728, ex c/c=204447587958, inc c/c=204447587958)
```

This part displays the execution time of each operator (including the execution time of filtering and projection, if any), CPU usage, and buffer usage.

– Operator execution information

```
4 --Vector Aggregate
   dn_6001_6002 (actual time=54076.307..54076.308 rows=1 loops=1) (projection time=65.581)
   dn_6003_6004 (actual time=68597.121..68597.122 rows=1 loops=1) (projection time=64.801)
```

The execution information of each operator consists of three parts:

- **dn_6001_6002/dn_6003_6004** indicates the information about the execution node. The information in the brackets is the actual execution information.
- **actual time** indicates the actual execution time. The first number indicates the duration from the time when the operator is executed to the time when the first data record is output. The second number indicates the total execution time of all data records.
- **rows** indicates the number of output data rows of the operator.

- **loops** indicates the number of execution times of the operator. Note that for a partitioned table, scan on each partition is counted as a scan. Scan on a new partition is counted as a new scan.
- CPU information

```
dn_6001_6002 (CPU: ex c/r=44, ex row=29999729, ex cyc=1346672758, inc cyc=162228870602)
dn_6003_6004 (CPU: ex c/r=44, ex row=29999728, ex cyc=1343711688, inc cyc=205791299646)
```

Each operator execution process has CPU information. **cyc** indicates the number of CPU cycles, and **ex cyc** indicates the number of cycles of the current operator, excluding its subnodes. **inc cyc** indicates the number of cycles, including subnodes, **ex row** indicates the number of data rows output by the current operator, and **ex c/r** indicates the mean of **ex cyc** and **ex row**.

- Buffer information

```
dn_6001_6002 (Buffers: shared hit=1725 dirtied=117)
dn_6003_6004 (Buffers: shared hit=1723 dirtied=117)
```

Buffers indicates the buffer information, including the read and write operations on shared blocks and temporary blocks.

Shared blocks contain tables and indexes, and temporary blocks are disk blocks used in sorting and materialization. The number of blocks displayed on the upper-layer node contains the number of blocks used by all its subnodes.

- Disk cache information (supported only by V3 tables or foreign tables with decoupled storage and compute and colversion set to 3.0 in 9.1.0.100 or later)

```
dn_6001_6002 (Disk Cache: hit=145611 miss=47 error=47 errorCode=152 scanBytes=18325.02MB remoteReadBytes=959.00MB loadTime=13109.5)
(Column 3.0: preloadStep=20 preloadSubmitTime=1230.6 preloadWaitTime=1724.9 preloadWaitCount=28)
(OBS I/O: count=349 averageRTT=238.7 averageLatency=233.0)
dn_6003_6004 (Disk Cache: hit=140641 miss=45 error=45 errorCode=152 scanBytes=17508.64MB remoteReadBytes=902.00MB loadTime=13271.7)
(Column 3.0: preloadStep=20 preloadSubmitTime=1202.9 preloadWaitTime=3061.5 preloadWaitCount=20)
(OBS I/O: count=327 averageRTT=231.5 averageLatency=226.3 latencyGtlis=1)
dn_6005_6006 (Disk Cache: hit=139736 miss=47 error=47 errorCode=152 scanBytes=17494.91MB remoteReadBytes=909.00MB loadTime=12752.5)
(Column 3.0: preloadStep=20 preloadSubmitTime=1169.4 preloadWaitTime=2723.4 preloadWaitCount=13)
(OBS I/O: count=327 averageRTT=194.3 averageLatency=188.8 latencyGtlis=3)
```

Disk Cache indicates the hit information and data read information of the disk cache. (supported by V3 tables or foreign tables with storage and compute decoupled)

miss indicates the number of disk cache misses. **hit** indicates the number of disk cache hits. For details about **errorCode**, see

disk_cache_error_code in [Table 18-162](#). **error** indicates the number of times **errorCode** is generated. **scanBytes** indicates the amount of data queried by scan, **remoteReadBytes** indicates the amount of data read on OBS, and **loadTime** indicates the time for loading data from the disk cache. To improve OBS efficiency, adjacent request blocks are combined. Alternatively, the minimum granularity for writing requests to the disk cache is block (1 MB by default). As a result, the value of **scanBytes** may be less than that of **remoteReadBytes**.

Column 3.0: prefetch parameters and prefetch process information of V3 tables with storage and compute decoupled. (This parameter is supported only by V3 tables with storage and compute decoupled and is displayed after prefetch parameters are enabled.)

preloadStep indicates the prefetch step, **preloadSubmitTime** indicates the I/O request submission time in the prefetch process, **preloadWaitTime** indicates the I/O request waiting time in the prefetch

process, and **preloadWaitCount** indicates the number of waiting I/O requests in the prefetch process.

OBS I/O indicates details about an OBS I/O request. (supported by V3 tables or foreign tables with storage and compute decoupled)

count indicates the total number of OBS I/O requests. **averageRTT** indicates the average round trip time (RTT) of OBS I/O requests. The unit is μ s. **averageLatency** indicates the average latency of OBS I/O requests. The unit is μ s. **latencyGt1s** indicates the number of OBS I/O requests whose latency exceeds 1s. **latencyGt10s** indicates the number of OBS I/O requests whose latency exceeds 10s. **retryCount** indicates the total number of OBS I/O request retries. **rateLimitCount** indicates the total number of times that OBS I/O requests are under flow control.

7. User Define Profiling

```
----- User Define Profiling -----
Plan Node id: 1 Track name: coordinator get datanode connection
cn_5001 (time=9.306 total_calls=1 loops=1)
Plan Node id: 1 Track name: coordinator begin transaction
cn_5001 (time=0.002 total_calls=1 loops=1)
Plan Node id: 1 Track name: coordinator send command
cn_5001 (time=0.113 total_calls=3 loops=1)
Plan Node id: 1 Track name: coordinator get the first tuple
cn_5001 (time=0.091 total_calls=12 loops=1)
```

User-defined information, including the time when CNs and DN are connected, the time when DNs are connected, and some execution information at the storage layer.

8. Query Summary

```
===== Query Summary =====
-----
Datanode executor start time [dn_6005_6006, dn_6001_6002]: [0.360 ms,0.483 ms]
Datanode executor run time [dn_6001_6002, dn_6003_6004]: [0.008 ms,0.009 ms]
Datanode executor end time [dn_6003_6004, dn_6005_6006]: [0.036 ms,0.066 ms]
Remote query poll time: 2.649 ms, Deserialize time: 0.000 ms
System available mem: 1761280KB
Query Max mem: 1761280KB
Query estimated mem: 3328KB
Enqueue time: 0.030 ms
Coordinator executor start time: 0.083 ms
Coordinator executor run time: 13.044 ms
Coordinator executor end time: 0.034 ms
Parser runtime: 0.060 ms
Planner runtime: 0.539 ms
Query Id: 218706056932222840
Unique SQL Id: 2641724793
Total runtime: 13.906 ms
```

The total execution time and network traffic, including the maximum and minimum execution time in the initialization and end phases on each DN, initialization, execution, and time in the end phase on each CN, and the system available memory during the current statement execution, and statement estimation memory information.

- **DataNode executor start time:** DN executor start time. The format is [min_node_name, max_node_name]: [min_time, max_time].
- **DataNode executor run time:** DN executor running duration. The format is [min_node_name, max_node_name]: [min_time, max_time].
- **DataNode executor end time:** DN executor end time. The format is [min_node_name, max_node_name]: [min_time, max_time].

- **Remote query poll time:** poll waiting time for receiving results.
- **System available mem:** available system memory
- **Query Max mem:** maximum query memory
- **Enqueue time:** enqueueing time
- **Coordinator executor start time:** CN executor start time
- **Coordinator executor run time:** CN executor running duration
- **Coordinator executor end time:** CN executor end time
- **Parser runtime:** parser running duration
- **Planner runtime:** optimizer execution duration
- **Network traffic:** amount of data sent by the stream operator
- **Query Id:** query ID
- **Unique SQL ID:** constraint SQL ID
- **Total runtime:** total execution duration

NOTICE

- The difference between A-rows and E-rows shows the deviation between the optimizer estimation and actual execution. Generally, if the deviation is large, the plan generated by the optimizer cannot be trusted, and you need to modify the deviation value.
 - If the difference of the A-time values is large, it indicates that the operator computing skew (difference between execution time on DN) is large and that manual performance tuning is required. Generally, for two adjacent operators, the execution time of the upper-layer operator includes that of the lower-layer operator. However, if the upper-layer operator is a stream operator, its execution time may be less than that of the lower-layer operator, as there is no driving relationship between threads.
 - **Max Query Peak Memory** is often used to estimate the consumed memory of SQL statements, and is also used as an important basis for setting a memory parameter during SQL statement optimization. Generally, the output from **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** is provided for the input for further optimization.
-

16.4.3 Execution Plan Operator

Operator Introduction

In an SQL execution plan, each step indicates a database operator, also called an execution operator. In DWS, operators are the building blocks of data processing. By combining them effectively and optimizing their sequence and execution, you can significantly improve data processing efficiency.

There are scan operators, control operators, materialization operators, join operators, and other operators in DWS.

Scan Operators

A scan operator scans data in a table, processing one tuple at a time for the upper-layer node. It operates at the leaf node of the query plan tree and can scan tables, result sets, linked lists, and subquery results. The following table lists common scan operators.

Table 16-9 Scan operators

Operator	Description	Scenario
SeqScan	Sequential scanning	It is a basic operator used to scan physical tables in sequence, not an index-assisted scan.
IndexScan	Index scanning	Indexes are created for the attributes involved in selection conditions.
IndexOnlyScan	Obtaining a tuple from an index	The index column completely overwrites the result set column.
BitmapScan(BitmapIndexScan, BitmapHeapScan)	Obtaining a tuple using a bitmap	BitmapIndexScan uses indexes for attributes to scan data and returns a bitmap. BitmapHeapScan then uses this bitmap to retrieve tuples.
TidScan	Obtaining a tuple by tuple tid	1. WHERE conditions(like CTID = tid or CTID IN (tid1, tid2, ...)) ; 2. UPDATE/DELETE ... WHERE CURRENT OF cursor;
SubqueryScan	Subquery scanning	Another query plan tree (subplan) is used as the scanning object to scan tuples.
FunctionScan	Function scanning	FROM function_name
ValuesScan	Values linked list scanning	It scans the given tuple set in VALUES clauses.
ForeignScan	External table scanning	It queries external tables.
CteScan	CTE table scanning	It scans the subquery defined by the WITH clause in the SELECT query.

Join Operators

In relational algebra, a join operation is equivalent to a join operator. Take a simple example: joining two tables, t1 and t2. There are several types of joins, including inner join, left join, right join, full join, semi join, and anti join. These joins can be implemented using three methods: Nestloop, HashJoin, and MergeJoin.

Table 16-10 Join operators

Operator	Description	Scenario	Implementation Feature
NestLoop	Nested loop join, which is a brute force approach. It scans the inner table for each row.	Inner Join, Left Outer Join, Semi Join, Anti Join	It is used for queries that have a smaller subset connected. In a nested loop, the foreign table drives the internal table. Each row returned by the foreign table is retrieved from the internal table to find the matched row. Therefore, the result set returned by the entire query cannot be greater than 10,000. The table with a smaller subset returned is used as the foreign table. It is recommended that indexes be created for the join fields in the internal table.
MergeJoin	A merge join on ordered input sorts both the inner and outer tables, identifies the first and last matching rows, and then joins tuples at a time. Equi-join.	Inner Join, Left Outer Join, Right Outer Join, Full Outer Join, Semi Join, Anti Join	In a merge join, data in the two joined tables is sorted by join columns. Then, data is extracted from the two tables to a sorted table for matching. A merge join requires more resources for sorting and its performance is lower than that of a hash join. However, if the source data has been pre-sorted and no more sorting is needed during the merge join, its performance excels.
(Sonic) Hash Join	Hash join: The inner and outer tables use the join column's hash value to create a hash table. Matching values are then stored in the same bucket. The two ends of an equal join must be of the same type and support hash.	Inner Join, Left Outer Join, Right Outer Join, Full Outer Join, Semi Join, Anti Join	A hash Join is used for large tables. The optimizer creates a hash table in memory using the join key and the smaller table. It then scans the larger table and uses the hash table to quickly identify matching rows. While Sonic and non-Sonic hash joins have different internal structures, this does not impact the final result set.

Materialization Operators

Materialization operators are a class of nodes that can cache tuples. During execution, many extended physical operations can be performed only after all tuples are obtained, such as aggregation function operations and sorting without indexes. Materialization operators can cache all the tuples.

Table 16-11 Materialization operators

Operator	Description	Scenario
Material	Materialization	Cache the subnode result.
Sort	Sorting	ORDER BY clause, which is used for join, group, and set operations and works with Unique.
Group	Grouping	GROUP BY clause.

Operator	Description	Scenario
Agg	Aggregate functions. There are three types: • HashAggregate: hash aggregation operator, which is used for aggregation calculation in most scenarios and in the case of disordered data. • GroupAggregate: sorting aggregation operator, which is used for grouping and aggregation based on sorted or unsorted inputs. • Aggregate: common aggregation operator, which is used for simple aggregation without grouping.	1. Aggregate functions such as COUNT, SUM, AVG, MAX, and MIN. 2. DISTINCT clause. 3. UNION deduplication. 4. GROUP BY clause.
WindowAgg	Window functions	WINDOW clause.

Operator	Description	Scenario
Unique	Deduplication (with sorted lower-layer data)	1. DISTINCT clause. 2. UNION deduplication.
Hash	HashJoin auxiliary node	Constructs a hash table and use it together with Hash Join.
SetOp	Processing set operations	INTERSECT/INTERSECT ALL, EXCEPT/EXCEPT ALL
LockRows	Processing row-level locks	SELECT ... FOR SHARE/UPDATE

Control Operators

Control operators are a type of node that handles exceptional scenarios and executes custom workflows.

Table 16-12 Control operators

Operator	Description	Scenario
Result	Performing calculation directly	1. Table scanning is not included. 2. The INSERT statement contains only one VALUES clause.
ModifyTable	INSERT/UPDATE/DELETE upper-layer node	INSERT, UPDATE, and DELETE
Append	Appending	1. UNION(ALL) 2. Table inheritance
MergeAppend	Appending (ordered input)	1. UNION(ALL) 2. Table inheritance
RecursiveUnion	Processing the UNION subquery defined recursively in the WITH clause	WITH RECURSIVE... SELECT... statement
BitmapAnd	Bitmap logical AND operation	BitmapScan for multi-dimensional index scanning
BitmapOr	Bitmap logical OR operation	BitmapScan for multi-dimensional index scanning
Limit	Processing the LIMIT clause	OFFSET ... LIMIT ...

Other Operators

Other operators include Stream and RemoteQuery. There are three types of Stream operators: Gather stream, Broadcast stream, and Redistribute stream.

- Gather stream: Each source node sends its data to the target node for aggregation.
- Broadcast stream: A source node sends its data to N target nodes for calculation.
- Redistribute stream: Each source node calculates the hash value of its data based on the join condition, distributes the data based on the hash value, and sends the data to the corresponding target node.

Table 16-13 Other Operators

Operator	Description	Scenario
Stream	Multi-node data exchange	When a distributed query plan is executed, data is exchanged between nodes.
Partition Iterator	Partition iterator	Scans partitioned tables and iteratively scans each partition.
RowToVec	Rows-to-column conversion	Hybrid row-column.
DfsScan / DfsIndexScan	HDFS table (index) scanning	HDFS table scanning.

16.4.4 SQL Tuning Process

You can analyze slow SQL statements to optimize them.

Procedure

- Step 1** Collect all table statistics associated with the SQL statements. In a database, statistics indicate the source data of a plan generated by a planner. If statistics are unavailable or out of date, the execution plan may seriously deteriorate, leading to low performance. According to past experience, about 10% performance problem occurred because no statistics are collected. For details, see [Updating Statistics](#).
- Step 2** [Review and modify the table definition.](#)
- Step 3** Generally, some SQL statements can be converted to its equivalent statements in all or certain scenarios by rewriting queries. SQL statements are simpler after they are rewritten. Some execution steps can be simplified to improve the performance.

The query rewriting method is universal in all databases. [SQL Statement Rewriting Rules](#) describes several optimization methods by rewriting SQL statements.

- Step 4** View the execution plan to find out the cause. If the SQL statements have been running for a long period of time and not ended, run the **EXPLAIN** command to view the execution plan and then locate the fault. If the SQL statement has been executed, run the **EXPLAIN ANALYZE** or **EXPLAIN PERFORMANCE** command to check the execution plan and actual running situation and then accurately locate the fault. For details about the execution plan, see [SQL Execution Plan](#).
- Step 5** For details about **EXPLAIN** or **EXPLAIN PERFORMANCE**, the reason why SQL statements are slowly located, and how to solve this problem, see [Advanced SQL Tuning](#).
- Step 6** Specify a join order; join, stream, or scan operations; number of rows in a result; or redistribution skew information to optimize an execution plan, improving query performance. For details, see [Hint-based Tuning](#).
- Step 7** To maintain high database performance, you are advised to perform [Routinely Maintaining Tables](#) and [Routinely Recreating an Index](#).
- Step 8** (Optional) Improve performance by using operators if resources are sufficient in DWS. For details, see [SMP Parallel Execution](#).

----End

16.4.5 Updating Statistics

In a database, statistics indicate the source data of a plan generated by a planner. If statistics are unavailable or out of date, the execution plan may seriously deteriorate, leading to low performance.

Scenario

The **ANALYZE** statement collects statistics on database table contents. These statistics will be stored in the **PG_STATISTIC** system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch **INSERT** and **DELETE** operations, you are advised to run the **ANALYZE** statement on the table or the entire database to update statistics. By default, 30,000 rows of statistics are sampled. That is, the default value of the GUC parameter **default_statistics_target** is **100**. If the total number of rows in the table exceeds 1,600,000, you are advised to set **default_statistics_target** to **-2**, indicating that 2% of the statistics are collected.

For an intermediate table generated during the execution of scripts or stored procedures in batch, you also need to run the **ANALYZE** statement.

If there are multiple inter-related columns in a table and the conditions or grouping operations based on these columns are involved in the query, collect statistics about these columns so that the query optimizer can accurately estimate the number of rows and generate an effective execution plan.

Generating Statistics

- Update statistics on a single table.
`ANALYZE tablename;`
- Update the statistics of the entire database.
`ANALYZE;`
- Collect statistics from multiple columns.
 - Collect statistics on the **column_1** and **column_2** columns of the **tablename** table.
`ANALYZE tablename ((column_1, column_2));`
 - --Add declarations for the **column_1** and **column_2** columns of the **tablename** table.
`ALTER TABLE tablename ADD STATISTICS ((column_1, column_2));`
 - Collect the statistics of a single column and statistics of multiple declared columns.
`ANALYZE tablename;`
 - Delete the statistics of **column_1** and **column_2** in the **tablename** table or their declarations.
`ALTER TABLE tablename DELETE STATISTICS ((column_1, column_2));`

NOTICE

- After the statistics are declared for multiple columns by running the **ALTER TABLE *Tablename* ADD STATISTICS** statement, the system collects the statistics about these columns next time **ANALYZE** is performed on the table or the entire database. To collect the statistics, run the **ANALYZE** statement.
- Use **EXPLAIN** to show the execution plan of each SQL statement. If **rows=10** (the default value, probably indicating the table has not been analyzed) is displayed in the **SEQ SCAN** output of a table, run the **ANALYZE** statement for this table.

Improving the Quality of Statistics

ANALYZE samples data from a table based on the random sampling algorithm and calculates table data features based on the samples. The number of samples can be specified by the **default_statistics_target** parameter. The value of **default_statistics_target** ranges from -100 to 10000 and the default value is 100.

- If the value of **default_statistics_target** is greater than 0, the number of samples is 300 x **default_statistics_target**. This means a larger value of **default_statistics_target** indicates a larger number of samples, larger memory space occupied by samples, and longer time required for calculating statistics.
- If the value of **default_statistics_target** is smaller than 0, the number of samples is **default_statistics_target**/100 x Total number of rows in the table. A smaller value of **default_statistics_target** indicates a larger number of samples. If the value of **default_statistics_target** is smaller than 0, the sampled data is written to the disk. In this case, the samples do not occupy memory. However, the calculation still takes a long time because the sample size is too large.

When **default_statistics_target** is negative, the number of samples is calculated as **default_statistics_target** divided by 100, multiplied by the total number of rows in the table. This sampling mode is also known as percentage sampling.

Automatic Statistics Collection

When the **autoanalyze** parameter is turned on, the optimizer will automatically collect statistics if it finds that there are no statistics in the table or if the data changes exceed a certain threshold. This ensures that the optimizer has the information it needs to make precise decisions.

In a cost-based optimizer (CBO) model, statistics play a crucial role in determining whether a query plan is generated. Therefore, it is crucial to have timely and effective statistics.

- Table-level statistics are stored in **relpages** and **reltuples** of **pg_class**.
- Column-level statistics, stored in **pg_statistics** and accessible through the **pg_stats** view, provide information on the percentage of NULL values, percentage of distinct values, high-frequency MCV values, and histograms.

Collection condition: If there is a substantial change in data volume (default threshold is **10%**), indicating a shift in data characteristics, the system will initiate the collection of statistics again.

Overall policy: The system enables dynamic sampling to collect statistics promptly and polling sampling to ensure persistent statistics. To ensure fast query performance with response times in seconds, it is recommended to use manual sampling.

Basic Rules

Table 16-14 Typical sampling methods

Function	Description	Feature	Constraint
Auto sampling	After making significant changes to the data in a job, you need to manually run the ANALYZE command.	• In normal mode, statistics are stored in system catalogs and shared globally. A level-4 lock is applied, preventing concurrent operations on a table. • In light mode, statistics are stored in memory and shared globally. A level-1 lock is applied, allowing concurrent operations on a table. • In force mode, you can perform forcible sampling even when statistics are locked, in addition to the normal mode functionalities. Syntax: ANALYZE tablename; ANALYZE (light force) tablename;	N/A
Polling sampling	Background thread operates according to a threshold. Polling maintenance statistics	Only the normal mode is supported. Statistics are stored in system catalogs and shared. A level-4 lock is applied, preventing concurrent operations on a table. Related GUC parameters: • autovacuum • autovacuum_mode • autovacuum_analyze_threshold • autovacuum_analyze_scale_factor	Asynchronous polling triggering

Function	Description	Feature	Constraint
Dynamic sampling	Depending on the threshold, the query parsing process can take several dozen seconds. Real-time maintenance statistics	- In normal mode, statistics are stored in system catalogs and shared globally. A level-4 lock is applied, preventing concurrent operations on a table. - In light mode, statistics are stored in memory and shared globally. A level-1 lock is applied, allowing concurrent operations on a table. Related GUC parameters: - autoanalyze - autoanalyze_mode	Real-time triggering upon query In lightweight scenarios, persistence relies on polling sampling.
Forcible sampling	Uses SQL hints to forcefully gather statistics for each query.	Used in data feature-sensitive scenarios to ensure real-time and up-to-date query statistics. Usage: select /*+ lightanalyze (t1 1) */ from t1; (1: forcible sampling; 0: sampling disabled)	The SQL statement needs to be modified.
Collecting partition statistics	Collects incremental information by partition and combines it globally.	Used in ultra-large partitioned tables to ensure accurate query cost estimation after partition pruning.	This method takes up more storage space but provides greater accuracy.
Collecting statistics from multiple columns	Gather statistics from multiple columns.	Used to filter multiple columns simultaneously to ensure accurate query cost estimation.	You need to select target columns manually and use temporary tables.
Collecting expression statistics	Collects statistics on a column based on expression functions.	Used in batch expression filtering scenarios to ensure accurate query cost estimation.	Manual identification is required.

Function	Description	Feature	Constraint
Collecting expression index statistics	Automatically collects statistics for created expression indexes.	Used in the point query expression filtering scenario to ensure accurate query cost estimation.	Manual identification is required.
Freezing statistics	Freezes table-level statistics to prevent changes.	Used in scenarios where data features are extremely stable to prevent sampling and query plan changes. Used in scenarios where data features are highly variable to ensure sampling for each query. Parameter: table-level attribute analyze_mode	N/A
Modifying statistics	Directly modifies statistics after manual calculation.	Used to maintain a low sampling ratio with manual calibration. Usage: select approx_count_distinct(col_name) from table_name; alter table set (n_distinct=xxx)	N/A
Copying partition information	Copies statistics from old partitions to new ones.	Used for partitioned tables with minimal data feature changes to reduce statistics collection overhead.	N/A
Statistical information inference	Automatically calculates more accurate statistics based on existing data.	Controlled by the GUC parameter enable_extrapolation_stats .	N/A
Backing up and restoring statistics	Backs up statistics to an SQL statement using the EXPLAIN (STAT ON) command.	Used for scenario reproduction or statistics restoration.	Statistics are exported as SQL statements.

Scenarios and Strategies

The table below outlines typical data processing scenarios and the corresponding strategies for collecting statistics.

Table 16-15 Statistics collection strategies

Scenario	Description	Strategy
Incremental stream processing	Incremental data flow changes with no reasonable time for ANALYZE .	Enable dynamic sampling to automatically collect and share statistics globally.
Online batch processing (Data lake)	Data processing and querying occur concurrently, requiring stable queries.	Enable dynamic sampling or complete data processing and ANALYZE within a transaction. begin; truncate table or partition; copy/merge/insert overwrite ANALYZE (light) tablename; end;
Partition parallel processing	Concurrent data processing in different partitions	Enable dynamic or manual light sampling and collect statistics concurrently for the same table.
Flat-wide table scenario	Wide table with over 100 columns	1. Enable automatic predicate management for dynamic sampling. 2. Collect statistics only on the first <i>N</i> columns. 3. Set column-level participation in sampling based on common query predicates.
Large table scenario	Large data volume with changes not reaching the threshold Variable statistics	Lower the threshold for triggering dynamic sampling.
Feature-sensitive scenario	Changeable data features causing unstable query plans, requiring forcible collection.	1. Lower the threshold for triggering dynamic sampling. 2. Use the HINT mode in SQL statements for light dynamic sampling. 3. Clear and freeze statistics, re-collecting them for each query without sharing.
High-concurrency scenario	Concurrent queries (over 10) are performed on the same table, triggering dynamic sampling and resource usage.	1. Disable concurrency, and other queries use outdated statistics. 2. Generate the latest statistics before querying (under development).

Scenario	Description	Strategy
Streaming performance sensitivity	Stream processing with queries responded in seconds or high resource usage	Disable dynamic sampling at the table or SQL level and use background polling sampling.
Batch performance sensitivity	Batch processing with queries responded in seconds or high resource usage	Manually collect statistics during processing.

Resource Consumption

Table 16-16 Resource consumption

Category	Sub-Category	Description
CPU	Predicate column management	Automatically manage predicates and collect statistics only on queried columns. Manually mask non-predicate columns.
	Ultra-long column statistics	Data type that can be truncated, counting only the first 1,024 characters.
I/O	30,000 samples are collected by default.	Related to the number of columns, partitions, and small CUs, not table size.
Memory	Buffer usage	At most one slot in the cstore buffer is occupied.
	Memory zero copy	Directly calculate statistics from buffer samples without organizing into tuples.
	Memory adaptation	Configure the system to use temporary tables for sampling when memory is insufficient. Prevent temporary table creation triggered by queries using the analyze_stats_mode parameter.
	Memory size	Control maximum memory usage during ANALYZE with the maintenance_work_mem parameter. Exceeding memory limits results in data being written to disks or reduced samples.

Category	Sub-Category	Description
Lock	Level-4 lock	(Normal mode) Applied in distributed mode, conflicting with DDL , VACUUM , ANALYZE , and REINDEX but not with addition, deletion, or modification.
	Level-1 lock	(Light mode) Only local level-1 lock is supported, conflicting only with DDL statements.

Accuracy and Reliability

Table 16-17 Accuracy/Reliability

Accuracy/Reliability	Item	Description
Accuracy	Sampling size	Configurable to adapt to table size with the default_statistics_target parameter.
	Sampling randomness	- Optimize reservoir and range sampling with the analyze_sample_mode parameter. - Enhance randomness of random number calculation with the random_function_version parameter.
	Global sharing	Statistics can be shared across sessions and nodes.
	Modifying count broadcast	Background thread checks and broadcasts the global modification count in polling mode. The job thread can also directly broadcast the modification count by specifying the tuple_change_sync_threshold parameter. Cross-CN modification and query have minimal impact. The modification count is broadcast and synchronized in asynchronous mode.
	Adjusting the CU sampling ratio	Increase CU sampling ratio if the CU filling rate is low, using the cstore_cu_sample_ratio parameter.
	Stabilizing distinct values	Use the n_distinct parameter to stabilize distinct values after random sampling without increasing the sampling ratio.

Accuracy/ Reliability	Item	Description
	Statistical information calculation	Use the enable_extrapolation_stats parameter to calculate more accurate statistics based on old statistics during distortion estimation.
Reliability	CN fault	Dynamic sampling is unaffected by other CN faults, and statistics are not synchronized. Query quality on the current CN remains unaffected.
	CN restoration	Forcibly perform dynamic sampling and global synchronization during queries after CN recovery.
	DN fault	Dynamic sampling of the logical cluster is unaffected by faults in other logical clusters.

O&M Monitoring

DWS offers a comprehensive view of the **ANALYZE** running mode and different execution stages by adding comments after the **ANALYZE** command. This information is primarily presented through the following views:

- **query** column in the **pgxc_stat_activity** view
- **wait_status** column in the **pgxc_thread_wait_status** view

The format of the **ANALYZE** command is **--Action-RunMode-StatsMode-SyncMode**.

- Values and meanings of **Action**:
{ "begin", "finished", "lock FirstCN", "estimate rows", "statistics", "sample rows", "calc stats" };
begin: indicates the start of the process; **finished**: indicates the end of the process; **lock FirstCN**: applies a lock from the FirstCN; **estimate rows**: estimates the number of rows in the first phase; **statistics**: executes **ANALYZE** in the second phase; **sample rows**: collects samples in the second phase; **calc stats**: calculates statistics in the second phase.
- Values and meanings of **RunMode**:
{ "manual", "backend", "normal runtime", "light runtime", "light runtime inxact", "light estimate rows", "light manual" };
manual: indicates the manual mode; **backend**: indicates the background polling mode; **normal runtime**: indicates the normal dynamic sampling; **light runtime**: indicates the light dynamic sampling; **light runtime inxact**: indicates the light dynamic sampling in a transaction; **light estimate rows**: indicates the light estimation function only; **light manual**: indicates the manual light mode.
- Values and meanings of **StatsMode**:
{ "dynamic", "memory", "smptbl" };

dynamic: indicates adaptive selection of memory or temporary table placement samples; **memory**: uses only internal storage samples; **smptbl**: uses only temporary table placement samples.

- Values and meanings of **SyncMode**:

```
{"sync", "nosync"};
```

sync: Statistics are synchronized to all CNs; **nosync**: Statistics are not synchronized.

Example:

```
SELECT coorname,datid,datname,pid,username,application_name,query_id,query
FROM pgxc_stat_activity WHERE query like '%analyze%' and query not like '%application_name%';
   coorname | datid | datname | pid | username | application_name | query_id | query
-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----
coordinator1 | 15676 | postgres | 139919333779200 | test | gsql | 73183493944770822 | analyze t_1;
coordinator2 | 15676 | postgres | 140217336461056 | test | coordinator1 | 73183493944770822 | analyze
public.t_1;--push stats-manual-memory-sync
coordinator3 | 15676 | postgres | 139944245847808 | test | coordinator1 | 73183493944770822 | analyze
public.t_1;--push stats-manual-memory-sync
(3 rows)
```

Viewing Statistics

- Check the dynamically sampled memory statistics.

- Retrieve table-level memory statistics.

```
SELECT * FROM pv_runtime_relstats;
```

- Retrieve column-level memory statistics.

```
SELECT * FROM pv_runtime_attstats;
```

- Check the system catalog statistics.

- Check the table-level system catalog statistics.

```
select relname, relpages, reltuples from pg_class;
```

- Check the column-level system catalog statistics.

```
SELECT * FROM pg_stats;
```

- Check the latest time when statistics are collected.

Dynamic sampling stores statistics in memory without modifying the timestamp of the system catalog.

```
SELECT * FROM pg_object;
```

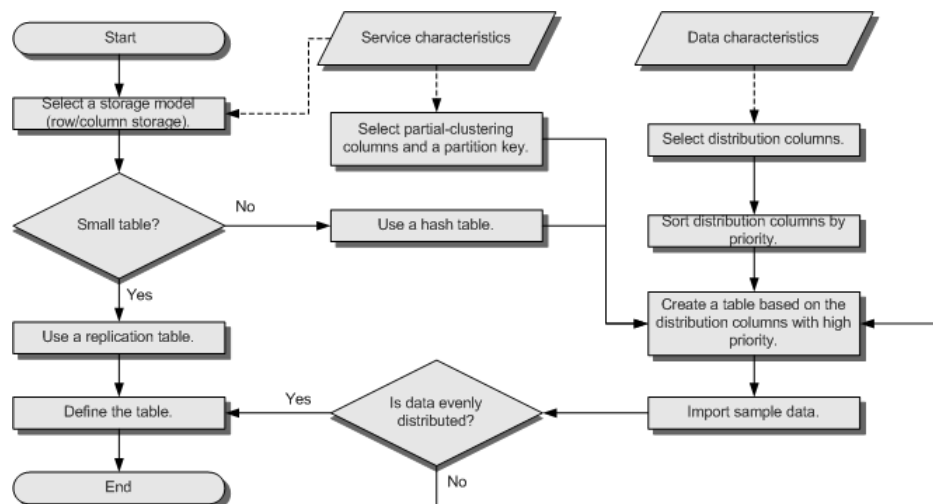
16.4.6 Reviewing and Modifying a Table Definition

In a distributed framework, data is distributed on DN. Data on one or more DN is stored on a physical storage device. To properly define a table, you must:

1. **Evenly distribute data on each DN** to avoid the available capacity decrease of a cluster caused by insufficient storage space of the storage device associated with a DN. Specifically, select a proper distribution key to avoid data skew.
2. **Evenly assign table scanning tasks on each DN** to avoid that a DN is overloaded by the table scanning tasks. Specifically, do not select columns in the equivalent filter of a base table as the distribution key.
3. **Reduce the data volume scanned** by using the partition pruning mechanism.
4. **Avoid the use of random I/O** by using clustering or partial clustering.
5. **Avoid data shuffle** to reduce the network pressure by selecting the **join-condition** column or **group by** column as the distribution column.

The distribution column is the core for defining a table. [Figure 16-5](#) shows the procedure of defining a table. The table definition is created during the database design and is reviewed and modified during SQL tuning.

Figure 16-5 Defining a table



16.4.7 Advanced SQL Tuning

16.4.7.1 SQL Self-Diagnosis

Performance issues may occur when you run the **INSERT/UPDATE/DELETE/SELECT/MERGE INTO** or **CREATE TABLE AS** statement. The product supports automatic performance diagnosis and saves related diagnosis information to [Real-time Top SQL](#). When [enable_resource_track](#) is set to **on**, the diagnosis information is dumped to [Historical Top SQL](#). You can query the **warning** column in the [GS_WLM_SESSION_STATISTICS](#), [GS_WLM_SESSION_HISTORY](#), and [GS_WLM_SESSION_INFO](#) views to obtain reference information for performance tuning.

- Alarms that can trigger SQL self-diagnosis depend on the settings of [resource_track_level](#).

When **resource_track_level** is set to **query**, you can diagnose alarms such as uncollected multi-column/single-column statistics, unpruned partitions, and failure of pushing down SQL statements. When **resource_track_level** is set to **perf** or **operator**, all alarms can be diagnosed.

- Whether a SQL plan will be diagnosed depends on the settings of [resource_track_cost](#).

A SQL plan will be diagnosed only if its execution cost is greater than **resource_track_cost**. You can use the **EXPLAIN** keyword to check the plan execution cost.

- When **EXPLAIN PERFORMANCE** or **EXPLAIN VERBOSE** is executed, SQL self-diagnosis information, except the ones without multi-column statistics, will be generated. For details, see [SQL Execution Plan](#).

Alarms Related to SQL Execution Performance

Currently, the following alarms on performance issues will be reported:

1. Statistics of a single column or multiple columns are not collected.

If statistics of a single column or multiple columns are not collected, an alarm is reported. To handle this alarm, you are advised to perform **ANALYZE** on related tables. For details, see [Updating Statistics](#) and [Optimizing Statistics](#).

If no statistics are collected for the OBS foreign table and HDFS foreign table in the query statement, an alarm indicating that statistics are not collected will be reported. Because the **ANALYZE** performance of the OBS foreign table and HDFS foreign table is poor, you are not advised to perform **ANALYZE** on these tables. Instead, you are advised to use the **ALTER FOREIGN TABLE** syntax to modify the **totalrows** attribute of the foreign table to correct the estimated number of rows.

Example alarms:

The statistics about a table are not collected.

```
Statistic Not Collect  
schema_test.t1
```

The statistics about a single column are not collected.

```
Statistic Not Collect  
schema_test.t2(c1)
```

The statistics about multiple columns are not collected.

```
Statistic Not Collect  
schema_test.t3((c1,c2))
```

The statistics about a single column and multiple columns are not collected.

```
Statistic Not Collect  
schema_test.t4(c1)  
schema_test.t5((c1,c2))
```

2. Partitions are not pruned.

When a partitioned table is queried, the partition is pruned based on the constraints on the partition key to improve the query performance. However, the partition table may not be pruned due to improper constraints, deteriorating the query performance. For details, see [Case: Rewriting SQL Statements and Eliminating Prune Interference](#).

3. SQL statements are not pushed down.

The cause details are displayed in the alarms. For details, see [Optimizing Statement Pushdown](#).

The potential causes for the pushdown failure are as follows:

- Caused by functions

The function name is displayed in the diagnosis information. The shippable attribute of the function determines whether the function can be pushed down. For details, see "CREATE FUNCTION" in the *Data Warehouse Service (DWS) SQL Syntax Reference*.

- Caused by syntax

The diagnosis information displays the syntax that causes the pushdown failure. For example, if the statement contains the **With Recursive**, **Distinct On**, or **row** expression and the return value is of the record type, an alarm is reported, indicating that the syntax does not support pushdown.

Example alarms:

```
SQL is not plan-shipping
  "enable_stream_operator" is off

SQL is not plan-shipping
  "Distinct On" can not be shipped

SQL is not plan-shipping
  "v_test_unshipping_log" is VIEW that will be treated as Record type can't be shipped
```

4. Vectorized plans are not supported.

For SQL statements that cannot use vectorized plans, detailed reasons why vectorized plans cannot be used are reported.

Common reasons are as follows:

- The target column contains functions whose return type is a set.
- The target column or query condition, the distribution key of the Stream operator, and the **Limit** and **Offset** clauses contain expressions that cannot be vectorized (such as geospatial types, array expressions, Row expressions, XML expressions, and functions whose parameters or return values contain the refcursor type).
- The **Group By** clause contains an array-equivalent judgment statement.
- **GC_FDW** and **LOG_FDW** do not support vectorization.
- The plan contains operators such as Cte Scan, Recursive Union, Merge Append, and Lock Rows.

Example alarms:

```
SQL is un-vectorized
  Function regexp_split_to_table that returns set is un-vectorized

SQL is un-vectorized
  Array expression is un-vectorized

SQL is un-vectorized
  Function array_agg is un-vectorized

SQL is un-vectorized
  RecursiveUnion is un-vectorized
```

5. In a hash join, the larger table is used as the inner table.

An alarm will be reported if the number of rows in the inner table reaches or exceeds 10 times of that in the foreign table, more than 100,000 inner-table rows are processed on each DN in average, and data has been flushed to disks. You can check the **query_plan** column in [GS_WLM_SESSION_HISTORY](#) to check whether hash joins are used. In this scenario, you need to adjust the sequence of the HashJoin internal and foreign tables. For details, see [Join Order Hints](#).

Example alarm:

```
Execute diagnostic information
PlanNode[7] Large Table is INNER in HashJoin "Vector Hash Aggregate"
```

In the preceding command, **7** indicates the operator whose ID is **7** in the **query_plan** column.

6. **nestloop** is used in a large-table equivalent join.

An alarm will be reported if nested loop is used in an equivalent join where more than 100,000 larger-table rows are processed on each DN in average. You can check the **query_plan** column of [GS_WLM_SESSION_HISTORY](#) to see if nested loop is used. In this scenario, you need to adjust the table join mode

and disable the NestLoop join mode between the current internal and foreign tables. For details, see [Join Operation Hints](#).

Example alarm:

```
Execute diagnostic information
PlanNode[5] Large Table with Equal-Condition use Nestloop"Nested Loop"
```

7. A large table is broadcasted.

An alarm will be reported if more than 100 thousand of rows are broadcasted on each DN in average. In this scenario, the broadcast operation of the Broadcast lower-layer operator needs to be disabled. For details, see [Stream Operation Hints](#).

Example alarm:

```
Execute diagnostic information
PlanNode[5] Large Table in Broadcast "Streaming(type: BROADCAST dop: 1/2)"
```

8. Data skew occurs.

An alarm will be reported if the number of rows processed on any DN exceeds 100 thousand, and the number of rows processed on a DN reaches or exceeds 10 times of that processed on another DN. Generally, this alarm is generated due to storage layer skew or computing layer skew. For details, see [Optimizing Data Skew](#).

Example alarm:

```
Execute diagnostic information
PlanNode[6] DataSkew:"Seq Scan", min_dn_tuples:0, max_dn_tuples:524288
```

9. The index is improper.

During base table scanning, an alarm is reported if the following conditions are met:

- For row-store tables:
 - When the index scanning is used, the ratio of the number of output lines to the number of scanned lines is greater than 1/1000 and the number of output lines is greater than 10,000.
 - When sequential scanning is used, the number of output lines to the number of scanned lines is less than 1/1000, the number of output lines is less than or equal to 10,000, and the number of scanned lines is greater than 10,000.
- For column-store tables:
 - When the index scanning is used, the ratio of the number of output lines to the number of scanned lines is greater than 1/10000 and the number of output lines is greater than 100.
 - When sequential scanning is used, the number of output lines to the number of scanned lines is less than 1/10,000, the number of output lines is less than or equal to 100, and the number of scanned lines is greater than 10,000.

For details, see [Tuning Operators](#). You can also refer to [Case: Creating an Appropriate Index](#) and [Case: Setting Partial Cluster Keys](#).

Example alarms:

```
Execute diagnostic information
PlanNode[4] Indexscan is not properly used:"Index Only Scan", output:524288, filtered:0,
rate:1.00000
PlanNode[5] Indexscan is ought to be used:"Seq Scan", output:1, filtered:524288, rate:0.00000
```

The diagnosis result is only a suggestion for the current SQL statement. You are advised to create an index only for frequently used filter criteria.

10. Estimation is inaccurate.

An alarm will be reported if the maximum number or the estimated maximum number of rows processed on a DN is over 100,000, and the larger number reaches or exceeds 10 times of the smaller one. In this scenario, you can refer to [Rows Hints](#) to correct the estimation on the number of rows, so that the optimizer can re-design the execution plan based on the correct number.

Example alarm:

```
Execute diagnostic information
PlanNode[5] Inaccurate Estimation-Rows: "Hash Join" A-Rows:0, E-Rows:52488
```

Constraints

1. An alarm contains a maximum of 2048 characters. If the length of an alarm exceeds this value (for example, a large number of long table names and column names are displayed in the alarm when their statistics are not collected), a warning instead of an alarm will be reported.
WARNING, "Planner issue report is truncated, the rest of planner issues will be skipped"
2. If a query statement contains the **Limit** operator, alarms of operators lower than **Limit** will not be reported.
3. For alarms about data skew and inaccurate estimation, only alarms on the lower-layer nodes in a plan tree will be reported. This is because the same alarms on the upper-level nodes may be triggered by problems on the lower-layer nodes. For example, if data skew occurs on the **Scan** node, data skew may also occur in operators (for example, **Hashagg**) at the upper layer.

16.4.7.2 Optimizing Statement Pushdown

Statement Pushdown

Currently, the DWS optimizer can use three methods to develop statement execution policies in the distributed framework: generating a statement pushdown plan, a distributed execution plan, or a distributed execution plan for sending statements.

- A statement pushdown plan pushes query statements from a CN down to DNs for execution and returns the execution results to the CN.
- In a distributed execution plan, a CN compiles and optimizes query statements, generates a plan tree, and then sends the plan tree to DNs for execution. After the statements have been executed, execution results will be returned to the CN.
- A distributed execution plan for sending statements pushes queries that can be pushed down (mostly base table scanning statements) to DNs for execution. Then, the plan obtains the intermediate results and sends them to the CN, on which the remaining queries are to be executed.

When sending statements through a distributed execution plan, DNs send numerous intermediate results to CNs. However, certain statements cannot be pushed down and must be executed on CNs, leading to performance bottlenecks

in bandwidth, storage, and computing. Therefore, you are not advised to use the query statements that only the third policy is applicable to.

Statements cannot be pushed down to DNs if they have **Functions That Do Not Support Pushdown** or **Syntax That Does Not Support Pushdown**. Generally, you can rewrite the execution statements to solve the problem.

Viewing Whether the Execution Plan Has Been Pushed Down to DNs

Perform the following procedure to quickly determine whether the execution plan can be pushed down to DNs:

- Step 1** Set the GUC parameter **enable_fast_query_shipping** to **off** to use the distributed framework policy for the query optimizer.

```
SET enable_fast_query_shipping = off;
```

- Step 2** View the execution plan.

If the execution plan contains Data Node Scan, the SQL statements cannot be pushed down to DNs. If the execution plan contains Streaming, the SQL statements can be pushed down to DNs.

For example:

```
select
count(ss.ss_sold_date_sk order by ss.ss_sold_date_sk)c1
from store_sales ss, store_returns sr
where
sr.sr_customer_sk = ss.ss_customer_sk;
```

The execution plan is as follows, which indicates that the SQL statement cannot be pushed down.

```
QUERY PLAN
-----
Aggregate
-> Hash Join
Hash Cond: (ss.ss_customer_sk = sr.sr_customer_sk)
-> Data Node Scan on store_sales "_REMOTE_TABLE_QUERY_"
Node/s: All datanodes
-> Hash
-> Data Node Scan on store_returns "_REMOTE_TABLE_QUERY_"
Node/s: All datanodes
(8 rows)
```

----End

Syntax That Does Not Support Pushdown

SQL syntax that does not support pushdown is described using the following table definition examples:

```
CREATE TABLE CUSTOMER1
(
  C_CUSTKEY    BIGINT NOT NULL
, C_NAME      VARCHAR(25) NOT NULL
, C_ADDRESS    VARCHAR(40) NOT NULL
, C_NATIONKEY  INT NOT NULL
, C_PHONE     CHAR(15) NOT NULL
, C_ACCTBAL   DECIMAL(15,2) NOT NULL
, C_MKTSEGMENT CHAR(10) NOT NULL
, C_COMMENT   VARCHAR(117) NOT NULL
)
```

```
DISTRIBUTE BY hash(C_CUSTKEY);  
CREATE TABLE test_stream(a int, b float);--float does not support redistribution.  
CREATE TABLE sal_emp ( c1 integer[] ) DISTRIBUTE BY replication;
```

- The **returning** statement cannot be pushed down.

```
explain update customer1 set C_NAME = 'a' returning c_name;  
QUERY PLAN
```

```
-----  
Update on customer1 (cost=0.00..0.00 rows=30 width=187)  
Node/s: All datanodes  
Node expr: c_custkey  
-> Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=187)  
Node/s: All datanodes  
(5 rows)
```

- If columns in **count(distinct expr)** do not support redistribution, they do not support pushdown.

```
explain verbose select count(distinct b) from test_stream;  
QUERY PLAN
```

```
----- Aggregate (cost=2.50..2.51 rows=1 width=8)  
Output: count(DISTINCT test_stream.b)  
-> Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=8)  
Output: test_stream.b  
Node/s: All datanodes  
Remote query: SELECT b FROM ONLY public.test_stream WHERE true  
(6 rows)
```

- Statements using **distinct on** cannot be pushed down.

```
explain verbose select distinct on (c_custkey) c_custkey from customer1 order by c_custkey;  
QUERY PLAN
```

```
----- Unique (cost=49.83..54.83 rows=30 width=8)  
Output: customer1.c_custkey  
-> Sort (cost=49.83..52.33 rows=30 width=8)  
Output: customer1.c_custkey  
Sort Key: customer1.c_custkey  
-> Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30  
width=8)  
Output: customer1.c_custkey  
Node/s: All datanodes  
Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true  
(9 rows)
```

- In a statement using **FULL JOIN**, if the column specified using **JOIN** does not support redistribution, the statement does not support pushdown.

```
explain select * from test_stream t1 full join test_stream t2 on t1.a=t2.b;  
QUERY PLAN
```

```
----- Hash Full Join (cost=0.38..0.82 rows=30  
width=24)  
Hash Cond: ((t1.a)::double precision = t2.b)  
-> Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30 width=12)  
Node/s: All datanodes  
-> Hash (cost=0.00..0.00 rows=30 width=12)  
-> Data Node Scan on test_stream "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=30  
width=12)  
Node/s: All datanodes  
(7 rows)
```

- A statement containing array expressions cannot be pushed down.

```
explain verbose select array[c_custkey,1] from customer1 order by c_custkey;
```

```
QUERY PLAN  
----- Sort (cost=49.83..52.33 rows=30 width=8)  
Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey  
Sort Key: customer1.c_custkey  
-> Data Node Scan on "_REMOTE_SORT_QUERY_" (cost=0.00..0.00 rows=30 width=8)  
Output: (ARRAY[customer1.c_custkey, 1::bigint]), customer1.c_custkey  
Node/s: All datanodes  
Remote query: SELECT ARRAY[c_custkey, 1::bigint], c_custkey FROM ONLY public.customer1  
WHERE true ORDER BY 2  
(7 rows)
```


- Subplans that are shared among multiple threads and cannot be pushed down.

```
postgres=# explain verbose select c_custkey in (select c_custkey from customer1) b from customer1;
QUERY PLAN
```

```
-----
Data Node Scan on customer1 "_REMOTE_TABLE_QUERY_" (cost=2.50..5.00 rows=1000 width=8)
```

```
Output: (hashed SubPlan 1)
```

```
Node/s: All datanodes
```

```
Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
```

```
SubPlan 1
```

```
-> Data Node Scan on customer "_REMOTE_TABLE_QUERY_" (cost=0.00..0.00 rows=1000
width=8)
```

```
Output: public.customer.c_custkey
```

```
Node/s: All datanodes
```

```
Remote query: SELECT c_custkey FROM ONLY public.customer1 WHERE true
```

```
(9 rows)
```

- The following table describes the scenarios where a statement containing **WITH RECURSIVE** cannot be pushed down in the current version, as well as the causes.

No.	Scenario	Cause of Not Supporting Pushdown
1	The query contains foreign tables or HDFS tables.	LOG: SQL can't be shipped, reason: RecursiveUnion contains HDFS Table or ForeignScan is not shippable (In this table, LOG describes the cause of not supporting pushdown.) In the current version, queries containing foreign tables or HDFS tables do not support pushdown.
2	Multiple Node Groups	LOG: SQL can't be shipped, reason: With-Recursive under multi-nodegroup scenario is not shippable In the current version, pushdown is supported only when all base tables are stored and computed in the same Node Group.
3	WITH recursive t_result AS (SELECT dm,sj_dm,name,1 as level FROM test_rec_part WHERE sj_dm > 10 UNION SELECT t2.dm,t2.sj_dm,t2.name ' > ' t1.name,t1.level+1 FROM t_result t1 JOIN test_rec_part t2 ON t2.sj_dm = t1.dm) SELECT * FROM t_result t;	LOG: SQL can't be shipped, reason: With-Recursive does not contain "ALL" to bind recursive & none-recursive branches ALL is not used for UNION . In this case, the return result is deduplicated.

No.	Scenario	Cause of Not Supporting Pushdown
4	<pre>WITH RECURSIVE x(id) AS (select count(1) from pg_class where oid=1247 UNION ALL SELECT id+1 FROM x WHERE id < 5), y(id) AS (select count(1) from pg_class where oid=1247 UNION ALL SELECT id+1 FROM x WHERE id < 10) SELECT y.*, x.* FROM y LEFT JOIN x USING (id) ORDER BY 1;</pre>	LOG: SQL can't be shipped, reason: With-Recursive contains system table is not shippable A base table contains the system catalog.
5	<pre>WITH RECURSIVE t(n) AS (VALUES (1) UNION ALL SELECT n+1 FROM t WHERE n < 100) SELECT sum(n) FROM t;</pre>	LOG: SQL can't be shipped, reason: With-Recursive contains only values rte is not shippable Only VALUES is used for scanning base tables. In this case, the statement can be executed on the CN, and DNs are unnecessary.
6	<pre>select a.ID,a.Name, (with recursive cte as (select ID, PID, NAME from b where b.ID = 1 union all select parent.ID,parent.PID,parent.NAME from cte as child join b as parent on child.pid=parent.id where child.ID = a.ID) select NAME from cte limit 1) cName from (select id, name, count(*) as cnt from a group by id,name) a order by 1,2;</pre>	LOG: SQL can't be shipped, reason: With-Recursive recursive term correlated only is not shippable The correlation conditions of correlated subqueries are only in the recursion part, and the non-recursion part has no correlation condition.
7	<pre>WITH recursive t_result AS (select * from(SELECT dm,sj_dm,name,1 as level FROM test_rec_part WHERE sj_dm < 10 order by dm limit 6 offset 2) UNION all SELECT t2.dm,t2.sj_dm,t2.name ' > ' t1.name,t1.level+1 FROM t_result t1 JOIN test_rec_part t2 ON t2.sj_dm = t1.dm) SELECT * FROM t_result t;</pre>	LOG: SQL can't be shipped, reason: With-Recursive contains conflict distribution in none-recursive(Replicate) recursive(Hash) The replicate plan is used for limit in the non-recursion part but the hash plan is used in the recursion part, resulting in conflicts.

No.	Scenario	Cause of Not Supporting Pushdown
8	<pre>with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from (with recursive cte as (select * from rec_tb4 where id<4 union all select h.id,h.parentID,h.name from rec_tb4 h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID) h inner join cte c on h.id=c.parentID) SELECT id ,parentID,name from cte order by parentID,1,2,3;</pre>	LOG: SQL can't be shipped, reason: Recursive CTE references recursive CTE "cte" recursive of multiple-layers are nested. That is, a recursive is nested in the recursion part of another recursive.

Functions That Do Not Support Pushdown

This module describes the variability of functions. The function variability in DWS is as follows:

- **IMMUTABLE**

Indicates that the function always returns the same result if the parameter values are the same.

- **STABLE**

Indicates that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter values, but that its result varies by SQL statements.

- **VOLATILE**

Indicates that the function value can change even within a single table scan, so no optimizations can be made.

The volatility of a function can be obtained by querying its **provolatile** column in **pg_proc**. The value **i** indicates immutable, **s** indicates stable, and **v** indicates volatile. The valid values of the **proshippable** column in **pg_proc** are **t**, **f**, and **NULL**. This column and the **provolatile** column together describe whether a function is pushed down.

- If the **provolatile** of a function is **i**, the function can be pushed down regardless of the value of **proshippable**.
- If the **provolatile** of a function is **s** or **v**, the function can be pushed only if the value of **proshippable** is **t**.
- CTEs containing random are not pushed down, because pushdown may lead to incorrect results.

For a UDF, you can specify the values of **provolatile** and **proshippable** during its creation. For details, see CREATE FUNCTION.

In scenarios where a function does not support pushdown, perform one of the following as required:

- If it is a system function, replace it with a functionally equivalent one.
- If it is a UDF function, check whether its **provolatile** and **proshippable** are correctly defined.

Example: UDF

Define a user-defined function that generates fixed output for a certain input as the **immutable** type.

Use the TPCDS sales information as an example. You need to define a function to obtain the discount information.

```
CREATE FUNCTION func_percent_2 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
VOLATILE;
```

Run the following statement:

```
SELECT func_percent_2(ss_sales_price, ss_list_price)
FROM store_sales;
```

The execution plan is as follows:

```
Data Node Scan on store_sales "REMOTE TABLE QUERY"
  Output: func_percent_2(store_sales.ss_sales_price, store_sales.ss_list_price)
  Remote query: SELECT ss_sales_price, ss_list_price FROM ONLY store_sales WHERE true
(3 rows)
```

func_percent_2 is not pushed down, and **ss_sales_price** and **ss_list_price** are executed on a CN. In this case, a large amount of resources on the CN is consumed, and the performance deteriorates as a result.

In this example, the function returns certain output when certain input is entered. Therefore, we can modify the function to the following one:

```
CREATE FUNCTION func_percent_1 (NUMERIC, NUMERIC) RETURNS NUMERIC
AS 'SELECT $1 / $2 WHERE $2 > 0.01'
LANGUAGE SQL
IMMUTABLE;
```

Run the following statement:

```
SELECT func_percent_1(ss_sales_price, ss_list_price)
FROM store_sales;
```

The execution plan is as follows:

```
Data Node Scan on "REMOTE_FQS_QUERY" (cost=0.00..0.00 rows=0 width=0)
  Output: (func_percent_1(store_sales.ss_sales_price, store_sales.ss_list_price))
  Node/s: All datanodes
  Remote query: SELECT public.func_percent_1(ss_sales_price, ss_list_price) AS func_percent_1 FROM public.store_sales
(4 rows)
```

func_percent_1 is pushed down to DNs for quicker execution. (In TPCDS 1000X, where three CNs and 18 DNs are used, the query efficiency is improved by over 100 times).

16.4.7.3 Optimizing Subqueries

What Is a Subquery

When an application runs a SQL statement to operate the database, a large number of subqueries are used because they are more clear than table join. Especially in complicated query statements, subqueries have more complete and independent semantics, which makes SQL statements clearer and easy to understand. Therefore, subqueries are widely used.

Subqueries are classified into subqueries and sublinks based on the location of subqueries in SQL statements.

- Subquery: corresponds to a scope table (RangeTblEntry) in the query parse tree. That is, a subquery is a **SELECT** statement following immediately after the **FROM** keyword.
- Sublink: corresponds to an expression in the query parsing tree. That is, a sublink is a statement in the **WHERE** or **ON** clause or in the target list.

In conclusion, a subquery is a scope table and a sublink is an expression in the query parsing tree. Sublinks can appear in constraints and expressions. Based on the implementation of sublinks in DWS, sublinks can be classified into the following types:

- exist_sublink: corresponding to the **EXIST** and **NOT EXIST** statements.
- any_sublink: corresponding to the **OP ANY(SELECT...)** statement. **OP** can be the **IN**, **<**, **>**, or **=** operator.
- all_sublink: corresponding to the **OP ALL(SELECT...)** statement. **OP** can be the **IN**, **<**, **>**, or **=** operator.
- rowcompare_sublink: corresponding to the **RECORD OP (SELECT...)** statement.
- expr_sublink: corresponding to the **(SELECT** with a single target list item) statement.
- array_sublink: corresponding to the **ARRAY(SELECT...)** statement.
- cte_sublink: corresponding to the **WITH(...)** statement.

NOTICE

When a subquery in a query statement is in a different execution cluster than the parent query and includes filter criteria related to the parent query, the filter criteria cannot be applied across clusters in logical cluster mode. This leads to additional operator overhead, potentially resulting in poorer performance compared to a single-cluster setup.

The sublinks commonly used in OLAP and HTAP are exist_sublink and any_sublink. The sublinks are pulled up by the optimization engine of DWS. Because of the flexible use of subqueries in SQL statements, complex subqueries may affect query performance. Subqueries are classified into non-correlated subqueries and correlated subqueries.

- **Non-correlated subquery**

The execution of a subquery is independent from any attribute of outer queries. In this way, a subquery can be executed before outer queries.

Example:

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
  select c2
  from t2
  where t2.c2 IN (2,3,4)
);
```

QUERY PLAN

Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Right Semi Join
Hash Cond: (t2.c2 = t1.c1)
-> Streaming(type: REDISTRIBUTE)
Spawn on: All datanodes
-> Seq Scan on t2
Filter: (c2 = ANY ('{2,3,4}'::integer[]))
-> Hash
-> Seq Scan on t1
(10 rows)

– Correlated subquery

The execution of a subquery depends on some attributes of outer queries which are used as **AND** conditions of the subquery. In the following example, **t1.c1** in the **t2.c1 = t1.c1** condition is a dependent attribute. Such a subquery depends on outer queries and needs to be executed once for each outer query.

Example:

```
select t1.c1,t1.c2
from t1
where t1.c1 in (
  select c2
  from t2
  where t2.c1 = t1.c1 AND t2.c2 in (2,3,4)
);
```

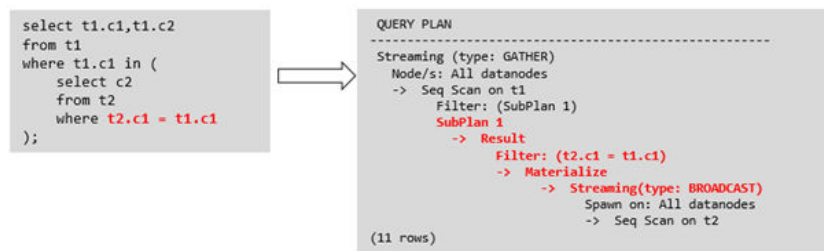
QUERY PLAN

Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on t1
Filter: (SubPlan 1)
SubPlan 1
-> Result
Filter: (t2.c1 = t1.c1)
-> Materialize
-> Streaming(type: BROADCAST)
Spawn on: All datanodes
-> Seq Scan on t2
Filter: (c2 = ANY ('{2,3,4}'::integer[]))
(12 rows)

DWS SubLink Optimization

A subquery is pulled up to join with tables in outer queries, preventing the subquery from being converted into the combination of a subplan and broadcast. You can run the **EXPLAIN** statement to check whether a subquery is converted into the combination of a subplan and broadcast.

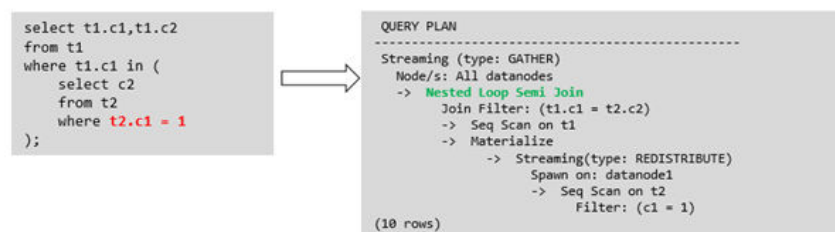
Example:



- **Sublink-release supported by DWS**

- Pulling up the **IN** sublink

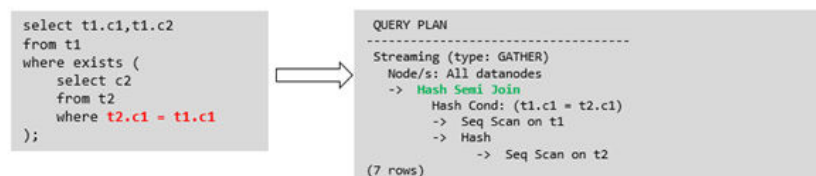
- The subquery cannot contain columns in the outer query (columns in more outer queries are allowed).
 - The subquery cannot contain volatile functions.



- Pulling up the **EXISTS** sublink

The **WHERE** clause must contain a column in the outer query. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

- The subquery must contain the **FROM** clause.
 - The subquery cannot contain the **WITH** clause.
 - The subquery cannot contain aggregate functions.
 - The subquery cannot contain a **SET**, **SORT**, **LIMIT**, **WindowAgg**, or **HAVING** operation.
 - The subquery cannot contain volatile functions.



- Pulling up an equivalent query containing aggregation functions

The **WHERE** condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using **AND**. Other parts of the subquery cannot contain the column. Other restrictions are as follows:

- The expression in the **WHERE** condition of the subquery must be table columns.
- After the **SELECT** keyword of the subquery, there must be only one output column. The output column must be an aggregate function (for example, **MAX**), and the parameter (for example, **t2.c2**) of the aggregate function cannot be columns of a table (for example, **t1**) in outer queries. The aggregate function cannot be **COUNT**.

For example, the following subquery can be pulled up:

```
select * from t1 where c1 >(
    select max(t2.c1) from t2 where t2.c1=t1.c1
);
```

The following subquery cannot be pulled up because the subquery has no aggregation function.

```
select * from t1 where c1 >(
    select t2.c1 from t2 where t2.c1=t1.c1
);
```

The following subquery cannot be pulled up because the subquery has two output columns:

```
select * from t1 where (c1,c2) >(
    select max(t2.c1),min(t2.c2) from t2 where t2.c1=t1.c1
);
```

- The subquery must be a **FROM** clause.
- The subquery cannot contain a **GROUP BY**, **HAVING**, or **SET** operation.
- The subquery can only be inner join.

For example, the following subquery can be pulled up:

```
select * from t1 where c1 >(
    select max(t2.c1) from t2 full join t3 on (t2.c2=t3.c2) where t2.c1=t1.c1
);
```

- The target list of the subquery cannot contain the function that returns a set.
- The **WHERE** condition of the subquery must contain a column from the outer query. Equivalence comparison must be performed between this column and related columns in tables of the subquery. These conditions must be connected using **AND**. Other parts of the subquery cannot contain the column. For example, the following subquery can be pulled up:

```
select * from t3 where t3.c1=(
    select t1.c1
    from t1 where c1 >(
        select max(t2.c1) from t2 where t2.c1=t1.c1
    ));
```

If another condition is added to the subquery in the previous example, the subquery cannot be pulled up because the subquery references to the column in the outer query. Example:

```
select * from t3 where t3.c1=(
    select t1.c1
    from t1 where c1 >(
        select max(t2.c1) from t2 where t2.c1=t1.c1 and t3.c1>t2.c2
    ));
```


- Pulling up a sublink in the **OR** clause

If the **WHERE** condition contains a **EXIST**-related sublink connected by **OR**,

for example,

```
select a, c from t1
where t1.a = (select avg(a) from t3 where t1.b = t3.b) or
exists (select * from t4 where t1.c = t4.c);
```

The procedure for promoting the OR clause of an EXIST-related subquery in an OR-ed join is as follows:

- Extract **opExpr** from the **OR** clause in the **WHERE** condition. The value is **t1.a = (select avg(a) from t3 where t1.b = t3.b)**.
- The **opExpr** contains a subquery. If the subquery can be pulled up, the subquery is rewritten as **select avg(a), t3.b from t3 group by t3.b**, generating the **NOT NULL** condition **t3.b is not null**. The **opExpr** is replaced with this **NOT NULL** condition. In this case, the SQL statement changes to:
- Extract the **EXISTS** sublink **exists (select * from t4 where t1.c = t4.c)** from the **OR** clause to check whether the sublink can be pulled up. If it can be pulled up, it is converted into **select t4.c from t4 group by t4.c**, generating the **NOT NULL** condition **t4.c is not null**. In this case, the SQL statement changes to:

```
select a, c
from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on (t1.a = avg
and t1.b = t3.b)
where t3.b is not null or exists (select * from t4 where t1.c = t4.c);
```

```
select a, c
from t1 left join (select avg(a) avg, t3.b from t3 group by t3.b) as t3 on (t1.a = avg and
t1.b = t3.b)
left join (select t4.c from t4 group by t4.c) where t3.b is not null or t4.c is not null;
```

```
select * from t1
where exists (
  select t2.c1 from t2
  where t2.c1 = t1.c1
) OR
exists (
  select t3.c1 from t3
  where t3.c1 = t1.c1
);
```



```
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Left Join
    Hash Cond: (t1.c1 = t3.c1)
    Filter: ((t2.c1 IS NOT NULL) OR (t3.c1 IS NOT NULL))
    -> Hash Left Join
        Hash Cond: (t1.c1 = t2.c1)
        -> Seq Scan on t1
        -> Hash
            -> HashAggregate
                Group By Key: t2.c1
                -> Seq Scan on t2
    -> Hash
        -> HashAggregate
            Group By Key: t3.c1
            -> Seq Scan on t3
(16 rows)
```

- **Sublink-release not supported by DWS**

Except the sublinks described above, all the other sublinks cannot be pulled up. In this case, a correlated subquery is planned as the combination of subplans and broadcast. As a result, if inner tables have a large amount of data, query performance may be poor.

If a correlated subquery joins with two tables in outer queries, the subquery cannot be pulled up. You need to change the parent query into a **WITH** clause and then perform the join.

Example:

```
select distinct t1.a, t2.a
from t1 left join t2 on t1.a=t2.a and not exists (select a,b from test1 where test1.a=t1.a and
test1.b=t2.a);
```

The parent query is changed into:

```
with temp as
(
    select * from (select t1.a as a, t2.a as b from t1 left join t2 on t1.a=t2.a)
)
select distinct a,b
from temp
where not exists (select a,b from test1 where temp.a=test1.a and temp.b=test1.b);
```

- The subquery (without **COUNT**) in the target list cannot be pulled up.

Example:

```
explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
from t1
where t1.c2 > 10;
```

The execution plan is as follows:

```
explain (costs off)
select (select c2 from t2 where t1.c1 = t2.c1) ssq, t1.c2
from t1
where t1.c2 > 10;
QUERY PLAN
```

```
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on t1
    Filter: (c2 > 10)
    SubPlan 1
        -> Result
            Filter: (t1.c1 = t2.c1)
            -> Materialize
                -> Streaming(type: BROADCAST)
                    Spawn on: All datanodes
                    -> Seq Scan on t2
```

(11 rows)

The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use a left outer join to join **t1** and **t2** so that the SSQ can return padding values when the condition **t1.c1=t2.c1** is not met.

NOTE

ScalarSubQuery (SSQ) and Correlated-ScalarSubQuery (CSSQ) are described as follows:

- SSQ: a sublink that returns only a single row and column scalar value
- CSSQ: an SSQ containing conditions

The preceding SQL statement can be changed into:

```
with ssq as
(
    select t2.c1, t2.c2 from t2
)
select ssq.c2, t1.c2
from t1 left join ssq on t1.c1 = ssq.c1
where t1.c2 > 10;
```

The execution plan after the change is as follows:

```
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Right Join
    Hash Cond: (t2.c1 = t1.c1)
    -> Seq Scan on t2
    -> Hash
        -> Seq Scan on t1
```

Filter: (c2 > 10)
(8 rows)

In the preceding example, the SSQ is pulled up to right join, preventing poor performance caused by the combination of a subplan and broadcast when the table (**T2**) in the subquery is too large.

- The subquery (with **COUNT**) in the target list cannot be pulled up.

Example:

```
select (select count(*) from t2 where t2.c1=t1.c1) cnt, t1.c1, t3.c1
from t1,t3
where t1.c1=t3.c1 order by cnt, t1.c1;
```

The execution plan is as follows:

```
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Sort
    Sort Key: ((SubPlan 1)), t1.c1
-> Hash Join
    Hash Cond: (t1.c1 = t3.c1)
-> Seq Scan on t1
-> Hash
    -> Seq Scan on t3
SubPlan 1
-> Aggregate
-> Result
    Filter: (t2.c1 = t1.c1)
-> Materialize
    -> Streaming(type: BROADCAST)
        Spawn on: All datanodes
    -> Seq Scan on t2
(17 rows)
```

The correlated subquery is displayed in the target list (query return list). Values need to be returned even if the condition **t1.c1=t2.c1** is not met. Therefore, use a left outer join to join **t1** and **t2** so that the SSQ can return padding values when the condition **t1.c1=t2.c1** is not met. However, **COUNT** is used, which requires that **0** is returned when the condition is not met. **case-when NULL then 0 else count(*)** can be used.

The preceding SQL statement can be changed into:

```
with ssq as
(
    select count(*) cnt, c1 from t2 group by c1
)
select case when
    ssq.cnt is null then 0
    else ssq.cnt
end cnt, t1.c1, t3.c1
from t1 left join ssq on ssq.c1 = t1.c1,t3
where t1.c1 = t3.c1
order by ssq.cnt, t1.c1;
```

The execution plan after the change is as follows:

```
QUERY PLAN
-----
Streaming (type: GATHER)
Node/s: All datanodes
-> Sort
    Sort Key: (count(*)), t1.c1
-> Hash Join
    Hash Cond: (t1.c1 = t3.c1)
-> Hash Left Join
    Hash Cond: (t1.c1 = t2.c1)
-> Seq Scan on t1
```

```
-> Hash
    -> HashAggregate
        Group By Key: t2.c1
        -> Seq Scan on t2
-> Hash
    -> Seq Scan on t3
(15 rows)
```

– Pulling up nonequivalent subqueries

Example:

```
select t1.c1, t1.c2
from t1
where t1.c1 = (select agg() from t2.c2 > t1.c2);
```

Nonequivalent subqueries cannot be pulled up. You can perform join twice (one CorrelationKey and one rownum self-join) to rewrite the statement.

You can rewrite the statement in either of the following ways:

■ Subquery rewriting

```
select t1.c1, t1.c2
from t1, (
    select t1.rowid, agg() aggref
    from t1,t2
    where t1.c2 > t2.c2 group by t1.rowid
) dt /* derived table */
where t1.rowid = dt.rowid AND t1.c1 = dt.aggref;
```

■ CTE rewriting

```
WITH dt as
(
    select t1.rowid, agg() aggref
    from t1,t2
    where t1.c2 > t2.c2 group by t1.rowid
)
select t1.c1, t1.c2
from t1, dt
where t1.rowid = dt.rowid AND
t1.c1 = dt.aggref;
```

NOTICE

- Currently, DWS does not have an effective way to provide globally unique row IDs for tables and intermediate result sets. Therefore, the rewriting is difficult. It is recommended that this issue is avoided at the service layer or by using **t1.xc_node_id + t1.ctid** to associate row IDs. However, the high repetition rate of **xc_node_id** leads to low association efficiency, and **xc_node_id+ctid** cannot be used as the join condition of hash join.
- If the AGG type is **COUNT(*)**, **0** is used for data padding if **CASE-WHEN** is not matched. If the type is not **COUNT(*)**, **NULL** is used.
- CTE rewriting works better by using share scan.

More Optimization Examples

1. Change the base table to a replication table and create an index on the filter column.

```
create table master_table (a int);
create table sub_table(a int, b int);
select a from master_table group by a having a in (select a from sub_table);
```

In this example, a correlated subquery is contained. To improve the query performance, you can change **sub_table** to a replication table and create an index on the **a** column.

2. Modify the **SELECT** statement, change the subquery to a **JOIN** relationship between the primary table and the parent query, or modify the subquery to improve the query performance. Ensure that the subquery to be used is semantically correct.

```
explain (costs off) select * from master_table as t1 where t1.a in (select t2.a from sub_table as t2 where t1.a = t2.b);
```

QUERY PLAN

Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on master_table t1
Filter: (SubPlan 1)
SubPlan 1
-> Result
Filter: (t1.a = t2.b)
-> Materialize
-> Streaming(type: BROADCAST)
Spawn on: All datanodes
-> Seq Scan on sub_table t2

(11 rows)

In the preceding example, a subplan exists in the plan. To delete the subplan, modify the statement as follows:

```
explain(costs off) select * from master_table as t1 where exists (select t2.a from sub_table as t2 where t1.a = t2.b and t1.a = t2.a);
```

QUERY PLAN

Streaming (type: GATHER)
Node/s: All datanodes
-> Hash Semi Join
Hash Cond: (t1.a = t2.b)
-> Seq Scan on master_table t1
-> Hash
-> Streaming(type: REDISTRIBUTE)
Spawn on: All datanodes
-> Seq Scan on sub_table t2

(9 rows)

In this way, the subplan is replaced by the semi-join between the two tables, greatly improving the execution efficiency.

16.4.7.4 Optimizing Statistics

What Is Statistic Optimization

DWS generates optimal execution plans based on the cost estimation. Optimizers need to estimate the number of data rows and the cost based on statistics collected using **ANALYZE**. Therefore, the statistics is vital for the estimation of the number of rows and cost. Global statistics are collected using **ANALYZE: relpages** and **reltuples** in the **pg_class** table; **stadistinct**, **stanullfrac**, **stanumbersN**, **stavaluesN**, and **histogram_bounds** in the **pg_statistic** table.

Example 1: Poor Query Performance Due to the Lack of Statistics

The query performance is often significantly impacted by the absence of statistics for tables or columns involved in the query.

The structure of the example table is as follows:

```
CREATE TABLE LINEITEM
(
  L_ORDERKEY      BIGINT      NOT NULL
, L_PARTKEY       BIGINT      NOT NULL
, L_SUPPKEY       BIGINT      NOT NULL
, L_LINENUMBER    BIGINT      NOT NULL
, L_QUANTITY       DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL
, L_DISCOUNT     DECIMAL(15,2) NOT NULL
, L_TAX           DECIMAL(15,2) NOT NULL
, L_RETURNFLAG    CHAR(1)     NOT NULL
, L_LINESTATUS    CHAR(1)     NOT NULL
, L_SHIPDATE      DATE        NOT NULL
, L_COMMITDATE    DATE        NOT NULL
, L_RECEIPTDATE   DATE        NOT NULL
, L_SHIPINSTRUCT  CHAR(25)    NOT NULL
, L_SHIPMODE      CHAR(10)    NOT NULL
, L_COMMENT       VARCHAR(44)  NOT NULL
) with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(L_ORDERKEY);

CREATE TABLE ORDERS
(
  O_ORDERKEY      BIGINT      NOT NULL
, O_CUSTKEY       BIGINT      NOT NULL
, O_ORDERSTATUS   CHAR(1)     NOT NULL
, O_TOTALPRICE    DECIMAL(15,2) NOT NULL
, O_ORDERDATE     DATE        NOT NULL
, O_ORDERPRIORITY CHAR(15)    NOT NULL
, O_CLERK         CHAR(15)    NOT NULL
, O_SHIPPRIORITY  BIGINT      NOT NULL
, O_COMMENT       VARCHAR(79)  NOT NULL
)with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(O_ORDERKEY);
```

The query statements are as follows:

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

You can perform the following operations to check whether **ANALYZE** has been executed on the tables or columns involved in the query to collect statistics.

1. Execute **EXPLAIN VERBOSE** to analyze the execution plan and check the warning information.

```
WARNING:Statistics in some tables or columns(public.lineitem(l_receiptdate,l_commitdate,l_orderkey,
l_suppkey), public.orders(o_orderstatus,o_orderkey)) are not collected.
HINT:Do analyze for them in order to generate optimized plan.
```

2. To determine if poor query performance was caused by a lack of statistics in certain tables or columns, check if the following information exists in the log file located in the **pg_log** directory.

```
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] LOG:Statistics in some tables
or columns(public.lineitem(l_receiptdate, l_commitdate,l_orderkey,
.l_suppkey), public.orders(o_orderstatus,o_orderkey)) are not collected.
2017-06-14 17:28:30.336 CST 140644024579856 20971684 [BACKEND] HINT:Do analyze for them in
order to generate optimized plan.
```

After confirming that **ANALYZE** has not been executed on the relevant tables or columns, you can execute **ANALYZE** on the tables or columns reported in the WARNING or logs to resolve the issue of slow query performance due to a lack of statistics

Example 2: Setting cost_param to Optimize Query Performance

For details, see [Case: Configuring cost_param for Better Query Performance](#).

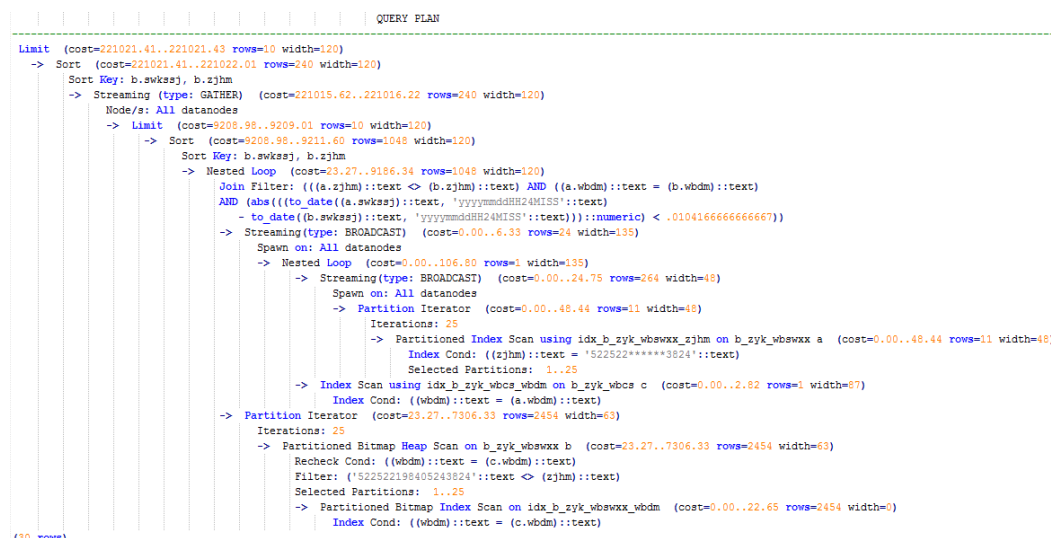
Example 3: Optimization is Not Accurate When Intermediate Results Exist in the Query Where JOIN Is Used for Multiple Tables

Symptom: Query the personnel who have checked in an Internet cafe within 15 minutes before and after the check-in of a specified person.

```
SELECT
C.WBM,
C.DZQH,
C.DZ,
B.ZJHM,
B.SWKSSJ,
B.XWSJ
FROM
b_zyk_wbswxx A,
b_zyk_wbswxx B,
b_zyk_wbcs C
WHERE
A.ZJHM = '522522*****3824'
AND A.WBDM = B.WBDM
AND A.WBDM = C.WBDM
AND abs(to_date(A.SWKSSJ,'yyyymmddHH24MISS') - to_date(B.SWKSSJ,'yyyymmddHH24MISS')) <
INTERVAL '15 MINUTES'
ORDER BY
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

Figure 16-6 shows the execution plan. This query takes about 12s.

Figure 16-6 Using an unlogged table (1)



Optimization analysis:

1. In the execution plan, index scan is used for node scanning, the **Join Filter** calculation in the external **NEST LOOP IN** statement consumes most of the query time, and the calculation uses the string addition and subtraction, and unequal-value comparison.
2. Use an unlogged table to record the Internet access time of the specified person. The start time and end time are processed during data insertion, and this reduces subsequent addition and subtraction operations.

```
//Create a temporary unlogged table.
```

```
CREATE UNLOGGED TABLE temp_tsw
```

```
(
  ZJHM      NVARCHAR2(18),
  WBDM      NVARCHAR2(14),
  SWKSSJ_START NVARCHAR2(14),
  SWKSSJ_END   NVARCHAR2(14),
  WBM        NVARCHAR2(70),
  DZQH       NVARCHAR2(6),
  DZ         NVARCHAR2(70),
  IPDZ       NVARCHAR2(39)
);
```

```
//Insert the Internet access record of the specified person, and process the start time and end time.
```

```
INSERT INTO
```

```
temp_tsw
```

```
SELECT
```

```
A.ZJHM,
```

```
A.WBDM,
```

```
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') - INTERVAL '15
```

```
MINUTES'),'yyyymmddHH24MISS'),
```

```
to_char((to_date(A.SWKSSJ,'yyyymmddHH24MISS') + INTERVAL '15
```

```
MINUTES'),'yyyymmddHH24MISS'),
```

```
B.WBM,B.DZQH,B.DZ,B.IPDZ
```

```
FROM
```

```
b_zyk_wbswxz A,
```

```
b_zyk_wbcs B
```

```
WHERE
```

```
A.ZJHM='522522*****3824' AND A.WBDM = B.WBDM
```

```
;
```

```
//Query the personnel who have check in an Internet cafe before and after 15 minutes of the check-in of the specified person. Convert their ID card number format to int8 in comparison.
```



```
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
order by
B.SWKSSJ,
B.ZJHM
limit 10 offset 0
;
```

The query takes about 7s. [Figure 16-7](#) shows the execution plan.

Figure 16-7 Using an unlogged table (2)

The query plan for Figure 16-7 (2) shows the following steps:

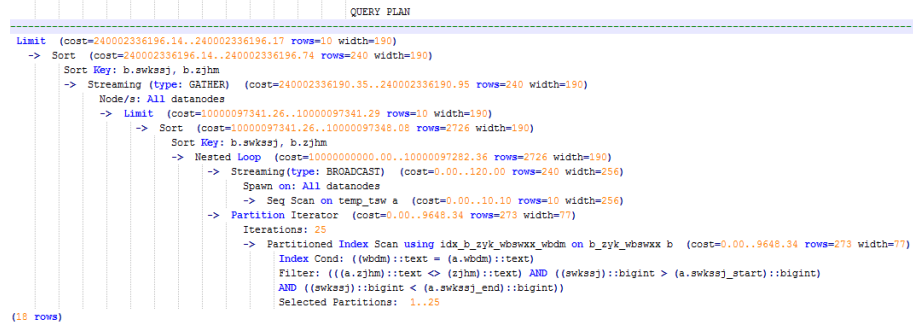
- Limit** (cost=13546726.90..13546726.92 rows=10 width=190)
 - Sort** (cost=13546726.90..13546727.50 rows=240 width=190)
 - Sort Key:** b.swkssj, b.zjhm
 - Streaming (type: GATHER)** (cost=13546721.11..13546721.71 rows=240 width=190)
 - Node/s: All datanodes**
 - Limit** (cost=564446.71..564446.74 rows=10 width=190)
 - Sort** (cost=564446.71..564453.53 rows=2726 width=190)
 - Sort Key:** b.swkssj, b.zjhm
 - Hash Join** (cost=533030.40..564387.81 rows=2726 width=190)
 - Hash Cond:** ((a.wbdm)::text = (b.wbdm)::text)
 - Join Filter:** (((a.zjhm)::text <> (b.zjhm)::text) AND ((b.swkssj)::bigint > (a.swkssj_start)::bigint) AND ((b.swkssj)::bigint < (a.swkssj_end)::bigint))
 - Streaming (type: BROADCAST)** (cost=0.00..120.00 rows=240 width=256)
 - Spawn on: All datanodes**
 - Seq Scan on temp_tsw a** (cost=0.00..10.10 rows=10 width=256)
 - Hash** (cost=465892.40..465892.40 rows=5371040 width=77)
 - Partition Iterator** (cost=0.00..465892.40 rows=5371040 width=77)
 - Iterations: 25**
 - Partitioned Seq Scan on b_zyk_wbswxx b** (cost=0.00..465892.40 rows=5371040 width=77)
 - Selected Partitions:** 1..25

3. In the previous plan, **Hash Join** has been executed, and a Hash table has been created for the large table **b_zyk_wbswxx**. The table contains large amounts of data, so the creation takes long time.

temp_tsw contains only hundreds of records, and an equal-value connection is created between **temp_tsw** and **b_zyk_wbswxx** using **wbdm** (the Internet cafe code). Therefore, if **JOIN** is changed to **NEST LOOP JOIN**, index scan can be used for node scanning, and the performance will be boosted.

4. Execute the following statement to change **JOIN** to **NEST LOOP JOIN**.
SET enable_hashjoin = off;

[Figure 16-8](#) shows the execution plan. The query takes about 3s.

Figure 16-8 Using an unlogged table (3)

5. Save the query result set in the unlogged table for paging display.

If paging display needs to be achieved on the upper-layer application page, change the **offset** value to determine the result set on the target page. In this way, the previous query statement will be executed every time after a page turning operation, which causes long response latency.

To resolve this problem, you are advised to use the unlogged table to save the result set.

```

//Create an unlogged table to save the result set.
CREATE UNLOGGED TABLE temp_result
(
WBM    NVARCHAR2(70),
DZQH   NVARCHAR2(6),
DZ     NVARCHAR2(70),
IPDZ   NVARCHAR2(39),
ZJHM   NVARCHAR2(18),
XM     NVARCHAR2(30),
SWKSSJ date,
XWSJ   date,
SWZDH  NVARCHAR2(32)
);

//Insert the result set to the unlogged table. The insertion takes about 3s.
INSERT INTO
temp_result
SELECT
A.WBM,
A.DZQH,
A.DZ,
A.IPDZ,
B.ZJHM,
B.XM,
to_date(B.SWKSSJ,'yyyymmddHH24MISS') as SWKSSJ,
to_date(B.XWSJ,'yyyymmddHH24MISS') as XWSJ,
B.SWZDH
FROM temp_tsw A,
b_zyk_wbswxx B
WHERE
A.ZJHM <> B.ZJHM
AND A.WBDM = B.WBDM
AND (B.SWKSSJ)::int8 > (A.swkssj_start)::int8
AND (B.SWKSSJ)::int8 < (A.swkssj_end)::int8
;

//Perform paging query on the result set. The paging query takes about 10 ms.
SELECT
*
FROM
temp_result
ORDER BY
SWKSSJ,

```

```
ZJHM
LIMIT 10 OFFSET 0;
```

CAUTION

Collecting global statistics using **ANALYZE** improves query performance.

If a performance problem occurs, you can use plan hint to adjust the query plan to the previous one. For details, see [Hint-based Tuning](#).

16.4.7.5 Tuning Operators

Background

A query goes through many steps to produce its final result. Often, the whole query slows down because one step takes too long. This slow step is called a bottleneck. To fix this, you can run the **EXPLAIN ANALYZE** or **PERFORMANCE** command to find the bottleneck.

In the example below, the **Hashagg** operator takes up 66% of the total time: $(51016 - 13535)/56476 \approx 66\%$. So, the **Hashagg** operator is the bottleneck. Start by optimizing this operator to improve performance.

| id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs |
|----|--|--------------------------|-------------|-------------|----------------------|----------|----------|---------|---------------|
| 1 | Row Adapter | [56476.397] | [100000000] | [237060] | [192B] | | | [20] | [2093222.75] |
| 2 | Vector Streaming (type: GATHER) | [55664.220] | [100000000] | [237060] | [243KB] | | | [20] | [2093222.75] |
| 3 | Vector Hash Aggregate | [55124.685, 55132.180] | [100000000] | [237060] | [29349KB, 29441KB] | [16MB] | [20, 20] | [20] | [20918406.50] |
| 4 | Vector Streaming (type: REDISTRIBUTE) | [51519.781, 52709.779] | [339164604] | [4856184] | [1219KB, 1219KB] | [1MB] | | [20] | [10461210.85] |
| 5 | Vector Hash Aggregate | [135875.525, 135815.424] | [339164604] | [4856184] | [721880KB, 746894KB] | [16MB] | [20, 20] | [20] | [10461210.85] |
| 6 | Vector Partition Iterator | [9035.202, 13565.884] | [970000000] | [915838097] | [9KB, 9KB] | [1MB] | | [20] | [10195891.68] |
| 7 | Partitioned CStore Scan on xujie_mp_day_energy_csv_1 | [9015.645, 13535.346] | [970000000] | [915838097] | [845KB, 845KB] | [1MB] | | [20] | [10195891.68] |

(7 rows)

Operator Tuning Example

Example 1: When scanning a base table with SeqScan, filtering large amounts of data, like in point queries or range scans, can be slow. Create an index on the condition column and use IndexScan to speed up the process.

```
explain (analyze on, costs off) select * from store_sales where ss_sold_date_sk = 2450944;
```

| id | operation | A-time | A-rows | Peak Memory | A-width |
|----|--------------------------|----------------------|--------|--------------|---------|
| 1 | Streaming (type: GATHER) | 3666.020 | 3360 | 195KB | |
| 2 | Seq Scan on store_sales | [3594.611, 3594.611] | 3360 | [34KB, 34KB] | |

Predicate Information (identified by plan id)

```
2 --Seq Scan on store_sales
  Filter: (ss_sold_date_sk = 2450944)
  Rows Removed by Filter: 4968936
```

```
create index idx on store_sales_row(ss_sold_date_sk);
```

```
CREATE INDEX
```

```
explain (analyze on, costs off) select * from store_sales_row where ss_sold_date_sk = 2450944;
```

| id | operation | A-time | A-rows | Peak Memory | A-width |
|----|---|------------------|--------|--------------|---------|
| 1 | Streaming (type: GATHER) | 81.524 | 3360 | 195KB | |
| 2 | Index Scan using idx on store_sales_row | [13.352, 13.352] | 3360 | [34KB, 34KB] | |

For instance, a full table scan returned 3,360 records but took 3.6 seconds. After indexing the **ss_sold_date_sk** column, the scan time dropped to 13 milliseconds.

Example 2: If **NestLoop** is chosen to join two tables and the row count is high, the join can be slow. In one case, **NestLoop** took 181 seconds. By setting

enable_mergejoin and **enable_nestloop** to **off** and letting the optimizer choose **HashJoin**, the join time was reduced to over 200 milliseconds.

```
postgres=# explain analyze select count(*) from store_sales ss, item i where ss.ss_item_sk = i.i_item_sk;
explain
          id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
-----
1 | --> Row Adapter | 184300.301 | 1 | 1 | 11KB | | | | | 0 | 48629179.77
2 | --> Vector Aggregate | 184300.280 | 1 | 1 | 161KB | | | | | 0 | 48629179.77
3 | --> Vector Streaming (type: GATHER) | 184300.196 | 4 | 4 | 168KB | | | | | 0 | 48629179.77
4 | --> Vector Aggregate | 1165575.384,184252.368 | 4 | 4 | 1140KB, 140KB | 1MB | | | | 0 | 48629179.61
5 | --> Vector Nest Loop (6,7) | 1162918.848,181438.162 | 2880404 | 2880404 | 174KB, 74KB | 1MB | | | | 0 | 48627379.35
6 | --> CStore Scan on store_sales ss | 115.460,16.293 | 2880404 | 2880404 | 490KB, 490KB | 1MB | | | | 4 | 16683.10
7 | --> Vector Materialize | 1118214.521,132478.454 | 12969211302 | 18000 | 8699KB, 900KB | 16MB | [8,8] | | 4 | 3890.00
8 | --> CStore Scan on item i | 0.234,0.243 | 18000 | 18000 | 476KB, 476KB | 1MB | | | | 4 | 3867.50
(8 rows)
```

```
postgres=# set enable_nestloop=off;
SET
postgres=# set enable_mergejoin=off;
SET
postgres=# explain analyze select count(*) fpostgres=# ales ss, item i where ss.ss_item_sk = i.i_item_sk;
explain
          id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
-----
1 | --> Row Adapter | 291.066 | 1 | 1 | 11KB | | | | | 0 | 32308.66
2 | --> Vector Aggregate | 291.052 | 1 | 1 | 181KB | | | | | 0 | 32308.66
3 | --> Vector Streaming (type: GATHER) | 290.973 | 4 | 4 | 188KB | | | | | 0 | 32308.66
4 | --> Vector Aggregate | 1220.792,234.532 | 4 | 4 | 1140KB, 140KB | 1MB | | | | 0 | 32308.50
5 | --> Vector Hash Join (6,7) | 1209.897,223.345 | 2880404 | 2880404 | 238KB, 243KB | 16MB | [8,8] | | 0 | 30508.24
6 | --> CStore Scan on store_sales ss | 13.132,13.717 | 2880404 | 2880404 | 490KB, 490KB | 1MB | | | | 4 | 16683.10
7 | --> Vector Materialize | 113.132,13.717 | 2880404 | 2880404 | 490KB, 490KB | 1MB | | | | 4 | 16683.10
(7 rows)
```

Example 3: **HashAgg** usually performs better. For large result sets, if **Sort** and **GroupAgg** are used, set **enable_sort** to **off**. **HashAgg** takes much longer time than **Sort** and **GroupAgg** used together.

```
postgres=# explain analyze select count(*) from store_sales group by ss_item_sk;
explain
          id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
-----
1 | --> Row Adapter | 1977.395 | 18000 | 17644 | 20KB | | | | | 4 | 92875.24
2 | --> Vector Streaming (type: GATHER) | 1973.621 | 18000 | 17644 | 194KB | | | | | 4 | 92875.24
3 | --> Vector Sort Aggregate | 1784.800,1883.243 | 18000 | 17644 | 273KB, 273KB | 1MB | | | | 4 | 92186.02
4 | --> Vector Aggregate | 1220.792,234.532 | 4 | 4 | 1140KB, 140KB | 1MB | | | | 0 | 32308.50
5 | --> Vector Hash Join (6,7) | 1209.897,223.345 | 2880404 | 2880404 | 238KB, 243KB | 16MB | [8,8] | | 0 | 30508.24
6 | --> CStore Scan on store_sales | 12.483,13.548 | 2880404 | 2880404 | 490KB, 490KB | 1MB | | | | 4 | 16683.10
(5 rows)
```

```
postgres=# set enable_sort=off;
SET
postgres=# explain analyze select count(*) from store_sales group by ss_item_sk;
explain
          id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs
-----
1 | --> Row Adapter | 838.218 | 18000 | 17644 | 20KB | | | | | 4 | 21016.93
2 | --> Vector Streaming (type: GATHER) | 834.264 | 18000 | 17644 | 228KB | | | | | 4 | 21016.93
3 | --> Vector Hash Aggregate | 585.017,758.204 | 18000 | 17644 | 262552KB, 262564KB | 16MB | [8,8] | | 4 | 20327.72
4 | --> CStore Scan on store_sales | 12.540,13.941 | 2880404 | 2880404 | 490KB, 490KB | 1MB | | | | 4 | 16683.10
(4 rows)
```

16.4.7.6 Optimizing Data Skew

Data skew breaks the balance among nodes in the distributed MPP architecture. If the amount of data stored or processed by a node is much greater than that by other nodes, the following problems may occur:

- Storage skew severely limits the system capacity. The skew on a single node hinders system storage utilization.
- Computing skew severely affects performance. The data to be processed on the skew node is much more than that on other nodes, deteriorating overall system performance.
- Data skew severely affects the scalability of the MPP architecture. During storage or computing, data with the same values is often placed on the same node. Therefore, even if you add nodes after a data skew occurs, the skew data (data with the same values) is still placed on the node and affects the system capacity or performance bottleneck.

DWS provides a complete solution for data skew, including storage and computing skew.

Data Skew in the Storage Layer

In the DWS database, data is distributed and stored on each DN. You can improve the query efficiency by using distributed execution. However, if data skew occurs, bottlenecks exist on some DNs during distribution execution, affecting the query

performance. This is because the distribution column is not properly selected. This can be solved by adjusting the distribution column.

Cause Analysis

- Step 1** Log in to the DWS console. On the **Clusters** page, locate the target cluster. In the **Operation** column of the target cluster, click **Monitoring Panel**. Choose **Monitoring > Node Monitoring**. Click the **Disks** tab to view the disk usage.

NOTE

Check the usage of each data disk. It is found that the usage is uneven among data disks. Generally, the difference between the highest and the lowest disk usage is small. If the difference exceeds 5%, data skew may occur.

- Step 2** Connect to the database and check the job operating status in the waiting view. It is found that the job waits for being processed by one or some DN.

```
SELECT wait_status, count(*) as cnt FROM pgxc_thread_wait_status WHERE wait_status not like '%cmd%'
AND wait_status not like '%none%' and wait_status not like '%quit%' group by 1 order by 2 desc;
```

- Step 3** The **explain performance** of the slow statement shows that the scan time and number of scan rows in the base table of each DN are unbalanced.

```
explain performance select avg(ss_wholesale_cost) from store_sales;
```

| operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-wi |
|---|--------------------|----------|---------|------------|----------------|----------|------|
| te | 2452.321 | 1 | 1 | | 11KB | | |
| te | 2452.238 | 1 | 1 | | 184KB | | |
| aming (type: GATHER) | 2452.010 | 4 | 4 | | 108KB | | |
| ggregate | [12.219, 2425.225] | 4 | 4 | | [183KB, 184KB] | 1MB | |
| r Partition Iterator | [12.139, 1189.187] | 22831616 | 2880404 | | [17KB, 17KB] | 1MB | |
| rtitioned CStore Scan on public.store_sales | [5.929, 1173.555] | 22831616 | 2880404 | | [482KB, 499KB] | 1MB | |

identified by plan id)

erator

can on public.store_sales

s: 1..7

- Time of scanning a base table: The fastest DN takes 5 ms, and the slowest DN takes 1173 ms.
- Data distribution: Some DNs have 22,831,616 rows and other DNs have no row, resulting in data skew.

```
5 --Vector Partition Iterator
  datanode1 (actual time=0.620..1189.187 rows=22831616 loops=1)
  datanode2 (actual time=14.346..14.346 rows=0 loops=1)
  datanode3 (actual time=14.424..14.424 rows=0 loops=1)
  datanode4 (actual time=12.139..12.139 rows=0 loops=1)
  datanode1 (CPU: ex c/r=1, ex row=22831616, ex c/c=40540820, inc c/c=3088825848)
  datanode2 (CPU: ex c/r=0, ex row=0, ex c/c=20852952, inc c/c=37297912)
  datanode3 (CPU: ex c/r=0, ex row=0, ex c/c=20005612, inc c/c=37501436)
  datanode4 (CPU: ex c/r=0, ex row=0, ex c/c=16147884, inc c/c=31560524)
6 --Partitioned CStore Scan on public.store_sales
  datanode1 (actual time=2.611..1173.555 rows=22831616 loops=7)
  datanode2 (actual time=6.327..6.327 rows=0 loops=7)
  datanode3 (actual time=6.732..6.732 rows=0 loops=7)
  datanode4 (actual time=5.929..5.929 rows=0 loops=7)
  datanode1 (Buffers: shared hit=1359)
  datanode2 (Buffers: shared hit=55)
  datanode3 (Buffers: shared hit=55)
  datanode4 (Buffers: shared hit=55)
  datanode1 (CPU: ex c/r=133, ex row=22831616, ex c/c=3048285028, inc c/c=3048285028)
  datanode2 (CPU: ex c/r=0, ex row=0, ex c/c=16444960, inc c/c=16444960)
  datanode3 (CPU: ex c/r=0, ex row=0, ex c/c=17495824, inc c/c=17495824)
  datanode4 (CPU: ex c/r=0, ex row=0, ex c/c=15412640, inc c/c=15412640)
```

- Step 4** You can detect data skew by using the skew check interface.

```
SELECT table_skewness('store_sales');
```

```
tpcds1xcpm=# select table_skewness('store_sales');
               table_skewness
```

```
-----
("datanode1      ",22831616,100.000%)
("datanode2      ",0,0.000%)
("datanode3      ",0,0.000%)
("datanode4      ",0,0.000%)
(4 rows)
```

```
SELECT table_distribution('public','store_sales');
```

```
tpcds1xcpm=# select table_distribution('public','store_sales');
               table_distribution
```

```
-----
(public,store_sales,datanode1,1011752960)
(public,store_sales,datanode4,33792000)
(public,store_sales,datanode2,32129024)
(public,store_sales,datanode3,33349632)
(4 rows)
```

----End

Solution – How to Find Skewed Tables:

1. If the number of tables in the database is less than 10,000, use the **PGXC_GET_TABLE_SKEWNESS** view to query data skew of all tables in the database.

```
SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
```
2. If the number of tables in the database is greater than 10,000, it may take a long time (hours) to query the entire database and calculate skew columns in the **PGXC_GET_TABLE_SKEWNESS** view. You are advised to perform the following operations by referring to the definition of the **PGXC_GET_TABLE_SKEWNESS** view:

- In 8.1.2 and earlier cluster versions, the **table_distribution()** function is used to optimize calculation by customizing output and reducing output columns. For example:

```
SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
FROM pg_catalog.pg_class c
INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pg_catalog.table_distribution() s ON s.schemaname = n.nspname AND s.tablename
= c.relname
INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND x.pclortype = 'H'
GROUP BY schemaname,tablename;
```

- For clusters of 8.1.3 and later versions, the **gs_table_distribution()** function can be used to check data skew of all tables in the database. The **gs_table_distribution()** function is better than the **table_distribution()** function when you query all tables in the database. In a large cluster with a large amount of data, use the **gs_table_distribution()** function.

```
SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
FROM pg_catalog.pg_class c
INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
INNER JOIN pg_catalog.gs_table_distribution() s ON s.schemaname = n.nspname AND
s.tablename = c.relname
INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND x.pclortype = 'H'
GROUP BY schemaname,tablename;
```

 NOTE

Run the following statement to query large tables:

```
SELECT schemaname||'.'||tablename as table, sum(dnsize) as size FROM
gs_table_distribution() group by 1 order by 2 desc limit 10;
```

Run the following statement to query the table skew rate:

```
WITH skew AS
(
    SELECT
        schemaname,
        tablename,
        pg_catalog.sum(dnsize) AS totalsize,
        pg_catalog.avg(dnsize) AS avgsz,
        pg_catalog.max(dnsize) AS maxsize,
        pg_catalog.min(dnsize) AS minsize,
        (pg_catalog.max(dnsize) - pg_catalog.min(dnsize)) AS skewsize,
        pg_catalog.stddev(dnsize) AS skewstddev
    FROM pg_catalog.pg_class c
    INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
    INNER JOIN pg_catalog.gs_table_distribution() s ON s.schemaname = n.nspname
    AND s.tablename = c.relname
    INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND x.pclortype IN('H',
    'N')
    GROUP BY schemaname,tablename
)
SELECT
    schemaname,
    tablename,
    totalsize,
    avgsz::numeric(1000),
    (maxsize/totalsize)::numeric(4,3) AS maxratio,
    (minsize/totalsize)::numeric(4,3) AS minratio,
    skewsize,
    (skewsize/avgsz)::numeric(4,3) AS skewratio,
    skewstddev::numeric(1000)
FROM skew
WHERE totalsize > 0;
```

Solution – How to Select a Distribution Key for a Table:

1. If the **distinct** value of the column is large and no obvious data skew occurs, you can define multiple columns as a distribution key.

View the size of **distinct**.

```
SELECT count(distinct column1) FROM table;
```

Check whether there are data skews:

```
SELECT count(*) cnt, column1 FROM table group by column1 order by cnt limit 100;
```

2. Select the columns where **JOIN** or **GROUP BY** statement is frequently used to reduce the use of **STREAM**.
3. It is advised against using these distribution key selection methods:
 - a. The default value of the distribution key (the first column) is used.
 - b. The distribution key is generated through the auto-increment of sequences.
 - c. The distribution key is generated using a random number. This method is recommended only when any column or any combination of two columns is skewed.

Data Skew in the Computing Layer

Even if data is balanced across nodes after you change the distribution key of a table, data skew may still occur during a query. If data skew occurs in the result

set of an operator on a DN, skew will also occur during the computing that involves the operator. Generally, this is caused by data redistribution during the execution.

During a query, JOIN keys and GROUP BY keys are not used as distribution columns. Data is redistributed among DNs based on the hash values of data on the keys. The redistribution is implemented using the Redistribute operator in an execution plan. Data skew in redistribution columns can lead to data skew during system operation. After the redistribution, some nodes will have much more data, process more data, and will have much lower performance than others.

In the following example, the **s** and **t** tables are joined, and **s.x** and **t.x** columns in the join condition are not their distribution keys. Table data is redistributed using the **REDISTRIBUTE** operator. Data skew occurs in the **s.x** column and not in the **t.x** column. The result set of the **Streaming** operator (**id** being **6**) on datanode2 has data three times that of other DNs and causes a skew.

```
select * from skew s, test t where s.x = t.x order by s.a limit 1;
```

| id | operation | A-time |
|----|----------------------------------|------------------------|
| 1 | -> Limit | 52622.382 |
| 2 | -> Streaming (type: GATHER) | 52622.374 |
| 3 | -> Limit | [30138.494, 52598.994] |
| 4 | -> Sort | [30138.486, 52598.986] |
| 5 | -> Hash Join (6,8) | [30127.013, 41483.275] |
| 6 | -> Streaming(type: REDISTRIBUTE) | [11365.110, 22024.845] |
| 7 | -> Seq Scan on public.skew s | [2019.168, 2175.369] |
| 8 | -> Hash | [2460.108, 2499.850] |
| 9 | -> Streaming(type: REDISTRIBUTE) | [1056.214, 1121.887] |
| 10 | -> Seq Scan on public.test t | [310.848, 325.569] |

```
6 --Streaming(type: REDISTRIBUTE)
  datanode1 (rows=5050368)
  datanode2 (rows=15276032)
  datanode3 (rows=5174272)
  datanode4 (rows=5219328)
```

Computing skew is more difficult to detect than storage skew. To solve computing skew, DWS provides the Runtime Load Balance Technology (RLBT) solution, controlled by the **skew_option** parameter. The RLBT solution addresses how to detect and solve data skew.

1. Detect data skew.

The solution first checks whether skew data exists in redistribution columns used for computing. RLBT can detect data skew based on statistics, specified hints, or rules.

– Detection based on statistics

Run the **ANALYZE** statement to collect statistics on tables. The optimizer will automatically identify skew data on redistribution keys based on the statistics and generate optimization plans for queries having potential skew. When the redistribution key has multiple columns, statistics information can be used for identification only when all columns belong to the same base table.

The statistics information can only provide the skew of the base table. If a column in the base table is skewed, or other columns have filtering conditions, or after the join of other tables, we cannot determine whether the skewed data still exists on the skewed column. If **skew_option** is **normal**, it indicates that the skew data still exists, and the base tables

will be optimized to solve skew. If **skew_option** is **lazy**, it indicates that no more skew data exists and the optimization will stop.

- Detection based on specified hints

The intermediate results of complex queries are difficult to estimate based on statistics. In this case, you can specify hints to provide the skew information based on which the optimizer optimizes queries. For details about the syntax of hints, see [Skew Hints](#).

- Detection based on rules

In a business intelligence (BI) system, a large number of SQL statements having outer joins (including left joins, right joins, and full joins) are generated, and many NULL values will be generated in empty columns that have no match for outer joins. If JOIN or GROUP BY operations are performed on the columns, data skew will occur. RLBT can automatically identify this scenario and generate an optimization plan for NULL value skew.

2. Solve computing skew.

Join and **Aggregate** operators are optimized to solve skew.

- **Join** optimization

Skew and non-skew data is separately processed. Details are as follows:

- When redistribution is required on both sides of a join:

Use **PART_REDISTRIBUTE_PART_ROUNDROBIN** on the side with skew. Specifically, perform round-robin on skew data and redistribution on non-skew data.

Use **PART_REDISTRIBUTE_PART_BROADCAST** on the side with no skew. Specifically, perform broadcast on skew data and redistribution on non-skew data.

- When redistribution is required on only one side of a join:

Use **PART_REDISTRIBUTE_PART_ROUNDROBIN** on the side where redistribution is required.

Use **PART_LOCAL_PART_BROADCAST** on the side where redistribution is not required. Specifically, perform broadcast on skew data and retain other data locally.

- When a table has **NULL** values padded:

Use **PART_REDISTRIBUTE_PART_LOCAL** on the table. Specifically, retain the **NULL** values locally and perform redistribution on other data.

In the example query, the **s.x** column contains skewed data and its value is **0**. The optimizer identifies the skew data in statistics and generates the following optimization plan:

| id | operation | A-time |
|----|---|-----------------------|
| 1 | -> Limit | 23642.049 |
| 2 | -> Streaming (type: GATHER) | 23642.041 |
| 3 | -> Limit | [23310.768,23618.021] |
| 4 | -> Sort | [23310.761,23618.012] |
| 5 | -> Hash Join (6,8) | [20898.341,21115.272] |
| 6 | -> Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN) | [7125.834,7472.111] |
| 7 | -> Seq Scan on public.skew s | [1837.079,1911.025] |
| 8 | -> Hash | [2612.484,2640.572] |
| 9 | -> Streaming(type: PART REDISTRIBUTE PART BROADCAST) | [1193.548,1297.894] |

```

10 |          -> Seq Scan on public.test t          | [314.343,328.707]

5 --Vector Hash Join (6,8)
  Hash Cond: s.x = t.x
  Skew Join Optimized by Statistic
6 --Streaming(type: PART REDISTRIBUTE PART ROUNDROBIN)
  datanode1 (rows=7635968)
  datanode2 (rows=7517184)
  datanode3 (rows=7748608)
  datanode4 (rows=7818240)

```

In the preceding execution plan, **Skew Join Optimized by Statistic** indicates that this is an optimized plan used for handling data skew. The **Statistic** keyword indicates that the plan optimization is based on statistics; **Hint** indicates that the optimization is based on hints; **Rule** indicates that the optimization is based on rules. In this plan, skew and non-skew data is separately processed. Non-skew data in the **s** table is redistributed based on its hash values, and skew data (whose value is **0**) is evenly distributed on all nodes in round-robin mode. In this way, data skew is solved.

To ensure result correctness, the **t** table also needs to be processed. In the **t** table, the data whose value is **0** (skew value in the **s.x** table) is broadcast and other data is redistributed based on its hash values.

In this way, data skew in JOIN operations is solved. The above result shows that the output of the **Streaming** operator (**id** being **6**) is balanced and the end-to-end performance of the query is doubled.

If the stream operator type in the execution plan is **HYBRID**, the stream mode varies depending on the skew data. The following plan is an example:

```

EXPLAIN (nodes OFF, costs OFF) SELECT COUNT(*) FROM skew_scol s, skew_scol1 s1 WHERE s.b =
s1.c;
QUERY PLAN

```

```

-----
id | operation
----
+-----
1 | -> Aggregate
2 | -> Streaming (type: GATHER)
3 | -> Aggregate
4 | -> Hash Join (5,7)
5 | -> Streaming(type: HYBRID)
6 | -> Seq Scan on skew_scol s
7 | -> Hash
8 | -> Streaming(type: HYBRID)
9 | -> Seq Scan on skew_scol1 s1

```

Predicate Information (identified by plan id)

```

-----
4 --Hash Join (5,7)
  Hash Cond: (s.b = s1.c)
  Skew Join Optimized by Statistic
5 --Streaming(type: HYBRID)
  Skew Filter: (b = 1)
  Skew Filter: (b = 0)
8 --Streaming(type: HYBRID)
  Skew Filter: (c = 0)
  Skew Filter: (c = 1)

```

Data 1 has skew in the **skew_scol** table. Perform **ROUNDROBIN** on skew data and **REDISTRIBUTE** on non-skew data.

Data 0 is the side with no skew in the **skew_scol** table. Perform **BROADCAST** on skew data and **REDISTRIBUTE** on non-skew data.

As shown in the preceding figure, the two stream types are **PART REDISTRIBUTE PART ROUNDROBIN** and **PART REDISTRIBUTE PART BROADCAST**. In this example, the stream type is **HYBRID**.

– **Aggregate optimization**

For aggregation, data on each DN is deduplicated based on the **GROUP BY** key and then redistributed. After the deduplication on DNs, the global occurrences of each value will not be greater than the number of DNs. Therefore, no serious data skew will occur. Take the following query as an example:

```
select c1, c2, c3, c4, c5, c6, c7, c8, c9, count(*) from t group by c1, c2, c3, c4, c5, c6, c7, c8, c9 limit 10;
```

The command output is as follows:

| id | operation | A-time | A-rows |
|--|----------------------------------|------------------------|----------|
| 1 | -> Streaming (type: GATHER) | 130621.783 | 12 |
| 2 | -> GroupAggregate | [85499.711,130432.341] | 12 |
| 3 | -> Sort | [85499.509,103145.632] | 36679237 |
| 4 | -> Streaming(type: REDISTRIBUTE) | [25668.897,85499.050] | 36679237 |
| 5 | -> Seq Scan on public.t | [9835.069,10416.388] | 36679237 |
| 4 --Streaming(type: REDISTRIBUTE)
datanode1 (rows=36678837)
datanode2 (rows=100)
datanode3 (rows=100)
datanode4 (rows=200) | | | |

A large amount of skew data exists. As a result, after data is redistributed based on its **GROUP BY** key, the data volume of datanode1 is hundreds of thousands of times that of others. After optimization, a GROUP BY operation is performed on the DN to deduplicate data. After redistribution, no data skew occurs.

| id | operation | A-time |
|----|----------------------------------|-----------------------|
| 1 | -> Streaming (type: GATHER) | 10961.337 |
| 2 | -> HashAggregate | [10953.014,10953.705] |
| 3 | -> HashAggregate | [10952.957,10953.632] |
| 4 | -> Streaming(type: REDISTRIBUTE) | [10952.859,10953.502] |
| 5 | -> HashAggregate | [10084.280,10947.139] |
| 6 | -> Seq Scan on public.t | [4757.031,5201.168] |

Predicate Information (identified by plan id)

| | |
|---|---|
| 3 | --HashAggregate
Skew Agg Optimized by Statistic |
| 4 | --Streaming(type: REDISTRIBUTE)
datanode1 (rows=17)
datanode2 (rows=8)
datanode3 (rows=8)
datanode4 (rows=14) |

Applicable scope

– **Join operator**

- **nest loop, merge join, and hash join** can be optimized.
- If skew data is on the left to the join, **inner join, left join, semi join, and anti join** are supported. If skew data is on the right to the join, **inner join, right join, right semi join, and right anti join** are supported.

- For an optimization plan generated based on statistics, the optimizer checks whether it is optimal by estimating its cost. Optimization plans based on hints or rules are forcibly generated.
- **Aggregate operator**
 - **array_agg**, **string_agg**, and **subplan in agg qual** cannot be optimized.
 - A plan generated based on statistics is affected by its cost, the **plan_mode_seed** parameter, and the **best_agg_plan** parameter. A plan generated based on hints or rules are not affected by them.

16.4.7.7 Proactive Preheating and Tuning of Disk Cache

This function is supported only in 9.1.0.200 or later.

Overview

In the storage-compute decoupling architecture, user data is stored in OBS to reduce storage costs. Each query generates network I/Os to retrieve data from OBS. To improve query speed, reduce storage costs, and minimize performance loss, the architecture provides disk cache capability. Pre-queried data is cached locally, enhancing performance.

The LRU2Q algorithm manages the disk cache with three queues: A1in, A1out, and Am. Data first enters A1in. If A1in is full (adjustable via a GUC parameter, default is 0.25 times the total queue size), data moves to A1out. Data enters Am only when hit in A1out. Am queue holds the hottest data.

For common queries, LRU2Q is sufficient. However, frequent joins of large and small tables can degrade performance if small tables are frequently evicted from A1in to A1out.

Tuning Syntax

A new tuning policy allows directly adding small table data to Am, ensuring it remains hot and reducing network I/Os during joins. The syntax formats are as follows:

1. Perform the actual query and add data directly to Am.

explain warmup hot select ...;

```
postgres=# explain warmup hot select * from orders1;
               QUERY PLAN
-----
----- Cache Data -----
 public.orders1;
    Read Cache Size : 39.095 MB
    Write Cache Size : 39.095 MB
    Avg Write Cache time : 5403.812ms
(7 rows)

postgres=# select * from pgac_disk_cache_all_stats;
 node_name | total_read | local_read | remote_read | hit_rate | cache_size | fill_rate | temp_file_size | a1in_size | a1out_size | am_size | a1in_fill_rate | a1out_fill_rate | am_fill_rate | fd | pin_block_count
-----
  dn_1     |      239  |      239   |           0 |    0.00 |      81312 |    1.55 |           0    |         0 |         0 |         0 |         0.00 |         0.00 |         0.00 | 2.07 |          0
  dn_2     |         0 |         0 |           0 |    0.00 |         0 |    0.00 |           0    |         0 |         0 |         0 |         0.00 |         0.00 |         0.00 | 0    |           0
(2 rows)
```

2. Query data in the sequence A1in > A1out > Am.

explain warmup select ...;

```
postgres=# explain warmup select * from orders1;
               QUERY PLAN
-----
----- Cache Data -----
 public.orders1;
    Read Cache Size : 39.095 MB
    Write Cache Size : 39.095 MB
    Avg Write Cache time : 22012.794ms
(7 rows)

postgres=# select * from pgac_disk_cache_all_stats;
 node_name | total_read | local_read | remote_read | hit_rate | cache_size | fill_rate | temp_file_size | a1in_size | a1out_size | am_size | a1in_fill_rate | a1out_fill_rate | am_fill_rate | fd | pin_block_count
-----
  dn_1     |      239  |      239   |           0 |    0.00 |      81312 |    1.55 |           0    |         0 |         0 |         0 |         0.00 |         0.00 |         0.00 | 2.07 |          0
  dn_2     |         0 |         0 |           0 |    0.00 |         0 |    0.00 |           0    |         0 |         0 |         0 |         0.00 |         0.00 |         0.00 | 0    |           0
(2 rows)
```

3. No actual query operation is performed, and the **explain** logic remains unchanged.
`explain select ...;`

16.4.7.8 SQL Statement Rewriting Rules

Based on the database SQL execution mechanism and a large number of practices, summarize finds that: using rules of a certain SQL statement, on the basis of the so that the correct test result, which can improve the SQL execution efficiency. You can comply with these rules to greatly improve service query efficiency.

- Replacing **UNION** with **UNION ALL**

UNION eliminates duplicate rows while merging two result sets but **UNION ALL** merges the two result sets without deduplication. Therefore, replace **UNION** with **UNION ALL** if you are sure that the two result sets do not contain duplicate rows based on the service logic.

- Adding **NOT NULL** to the join column

If there are many **NULL** values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.

- Converting **NOT IN** to **NOT EXISTS**

nestloop anti join must be used to implement **NOT IN**, and **Hash anti join** is required for **NOT EXISTS**. If no **NULL** value exists in the **JOIN** column, **NOT IN** is equivalent to **NOT EXISTS**. Therefore, if you are sure that no **NULL** value exists, you can convert **NOT IN** to **NOT EXISTS** to generate **hash joins** and to improve the query performance.

As shown in the following figure, the **t2.d2** column does not contain null values (it is set to **NOT NULL**) and **NOT EXISTS** is used for the query.

```
SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
```

The generated execution plan is as follows:

Figure 16-9 NOT EXISTS execution plan

```
id | operation
---+-----
1 | -> Streaming (type: GATHER)
2 | -> Hash Right Anti Join (3, 5)
3 | -> Streaming (type: REDISTRIBUTE)
4 | -> Seq Scan on t2
5 | -> Hash
6 | -> Seq Scan on t1

Predicate Information (identified by plan id)
-----
2 --Hash Right Anti Join (3, 5)
   Hash Cond: (t2.d2 = t1.c1)
(13 rows)
```

- Use **hashagg**.

If a plan involving groupAgg and SORT operations generated by the **GROUP BY** statement is poor in performance, you can set **work_mem** to a larger value to generate a **hashagg** plan, which does not require sorting and improves the performance.

- Replace functions with **CASE** statements
The DWS performance greatly deteriorates if a large number of functions are called. In this case, you can modify the pushdown functions to **CASE** statements.
- **Do not use functions or expressions for indexes.**
Using functions or expressions for indexes stops indexing. Instead, it enables scanning on the full table.
- Do not use **!=** or **<>** operators, **NULL**, **OR**, or implicit parameter conversion in **WHERE** clauses.
- **Split complex SQL statements.**
You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:
 - The same subquery is involved in multiple SQL statements of a task and the subquery contains large amounts of data.
 - Incorrect **Plan cost** causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.
 - Functions such as **substr** and **to_number** cause incorrect measures for subqueries containing large amounts of data.
 - **BROADCAST** subqueries are performed on large tables in multi-DN environment.

16.4.8 Configuring Optimizer Parameters

This section introduces key CN parameters that affect optimization of SQL statements in DWS. For details about the parameter configuration method, see [Configuring GUC Parameters](#).

Table 16-18 CN parameters

| Parameter/
Reference Value | Description |
|-------------------------------|---|
| enable_nestloop=on | <p>Specifies how the optimizer uses Nest Loop Join. If this parameter is set to on, the optimizer preferentially uses Nest Loop Join. If it is set to off, the optimizer preferentially uses other methods, if any.</p> <p>NOTE
To temporarily change the value of this parameter in the current database connection (that is, the current session), run the following SQL statement:
SET enable_nestloop to off;</p> <p>By default, this parameter is set to on. Change the value as required. Generally, nested loop join has the poorest performance among the three JOIN methods (nested loop join, merge join, and hash join). You are advised to set this parameter to off.</p> |

| Parameter/
Reference Value | Description |
|-------------------------------|---|
| enable_bitmapscan=on | <p>Specifies whether the optimizer uses bitmap scanning. If the value is on, bitmap scanning is used. If the value is off, it is not used.</p> <p>NOTE
If you only want to temporarily change the value of this parameter during the current database connection (that is, the current session), run the following SQL statements:
SET enable_bitmapscan to off;</p> <p>The bitmap scanning applies only in the query condition where a > 1 and b > 1 and indexes are created on columns a and b. During performance tuning, if the query performance is poor and bitmapscan operators are in the execution plan, set this parameter to off and check whether the performance is improved.</p> |
| enable_fast_query_shipping=on | <p>Specifies whether the optimizer uses a distribution framework. If the value is on, the execution plan is generated on both CNs and DN. If the value is off, the distribution framework is used, that is, the execution plan is generated on the CNs and then sent to DNs for execution.</p> <p>NOTE
To temporarily change the value of this parameter in the current database connection (that is, the current session), run the following SQL statement:
SET enable_fast_query_shipping to off;</p> |
| enable_hashagg=on | Specifies whether to enable the optimizer's use of Hash-aggregation plan types. |
| enable_hashjoin=on | Specifies whether to enable the optimizer's use of Hash-join plan types. |
| enable_mergejoin=on | Specifies whether to enable the optimizer's use of Hash-merge plan types. |
| enable_indexscan=on | Specifies whether to enable the optimizer's use of index-scan plan types. |
| enable_indexonlyscan=on | Specifies whether to enable the optimizer's use of index-only-scan plan types. |
| enable_seqscan=on | Specifies whether the optimizer uses bitmap scanning. It is impossible to suppress sequential scans entirely, but setting this variable to off allows the optimizer to preferentially choose other methods if available. |
| enable_sort=on | Specifies the optimizer sorts. It is impossible to fully suppress explicit sorts, but setting this variable to off allows the optimizer to preferentially choose other methods if available. |

| Parameter/
Reference Value | Description |
|---|--|
| enable_broadcast=on | Specifies whether enable the optimizer's use of data broadcast. In data broadcast, a large amount of data is transferred on the network. When the number of transmission nodes (stream) is large and the estimation is inaccurate, set this parameter to off and check whether the performance is improved. |
| enable_redistribute=on

This parameter is supported only by clusters of version 8.2.1.300 or later. | Controls the query optimizer's use of data transmission in local redistribute and split redistribute redistribution modes. This parameter corresponds to enable_broadcast . The optimizer may overestimate the cost of local broadcast and split broadcast . As a result, the optimizer selects the local redistribute or split redistribute redistribution plan. This may cause performance deterioration. Therefore, when the actual data volume of the network transmission node (stream) is small, you can set this parameter to off so that the optimizer preferentially selects the broadcast mode. Then you can check whether the performance is improved. |
| rewrite_rule | Specifies whether the optimizer enables a specific rewriting rule. |

16.4.9 Hint-based Tuning

16.4.9.1 Plan Hint Optimization

In plan hints, you can specify a join order, join, stream, and scan operations, the number of rows in a result, and redistribution skew information to tune an execution plan, improving query performance.

Function

Plan hints can be specified using the keywords such as **SELECT**, **INSERT**, **UPDATE**, **MERGE**, and **DELETE**, in the following format:

```
/*+ <plan hint> */
```

You can specify multiple hints for a query plan and separate them by spaces. A hint specified for a query plan does not apply to its subquery plans. To specify a hint for a subquery, add the hint following the keyword of this subquery.

For example:

```
select /*+ <plan_hint1> <plan_hint2> */ * from t1, (select /*+ <plan_hint3> */ from t2) where 1=1;
```

In the preceding command, *<plan_hint1>* and *<plan_hint2>* are the hints of a query, and *<plan_hint3>* is the hint of its subquery.

NOTICE

If a hint is specified in the **CREATE VIEW** statement, the hint will be applied each time this view is used.

If the random plan function is enabled (**plan_mode_seed** is set to a value other than 0), the specified hint will not be used.

Supported Hints

Currently, the following hints are supported:

- Join order hints (**leading**)
- Join operation hints, excluding the **semi join**, **anti join**, and **unique plan** hints
- Rows hints
- Stream operation hints
- Scan operation hints, supporting only **tablescan**, **indexscan**, and **indexonlyscan**
- Sublink name hints
- Skew hints, supporting only the skew in the redistribution involving Join or HashAgg
- Hint used for **Agg** distribution columns Only clusters of 8.1.3.100 and later versions support this function.
- Hint that disables subquery pull-up. Only clusters of 8.2.0 and later versions support this function.
- Configuration parameter hints. For details about supported parameters, see [Configuration Parameter Hints](#).

Precautions

- **Sort**, **Setop**, and **Subplan** hints are not supported.
- Hints do not support SMP or Node Groups.
- Hints cannot be used for the target table of the **INSERT** statement.

Examples

The following is the original plan and is used for comparing with the optimized ones:

```
explain
select i_product_name product_name
,i_item_sk item_sk
,s_store_name store_name
,s_zip store_zip
,ad2.ca_street_number c_street_number
,ad2.ca_street_name c_street_name
,ad2.ca_city c_city
,ad2.ca_zip c_zip
,count(*) cnt
,sum(ss_wholesale_cost) s1
,sum(ss_list_price) s2
,sum(ss_coupon_amt) s3
FROM   store_sales
,store_returns
```

```

,store
,customer
,promotion
,customer_address ad2
,item
WHERE ss_store_sk = s_store_sk AND
ss_customer_sk = c_customer_sk AND
ss_item_sk = i_item_sk and
ss_item_sk = sr_item_sk and
ss_ticket_number = sr_ticket_number and
c_current_addr_sk = ad2.ca_address_sk and
ss_promo_sk = p_promo_sk and
i_color in ('maroon','burnished','dim','steel','navajo','chocolate') and
i_current_price between 35 and 35 + 10 and
i_current_price between 35 + 1 and 35 + 15
group by i_product_name
,i_item_sk
,s_store_name
,s_zip
,ad2.ca_street_number
,ad2.ca_street_name
,ad2.ca_city
,ad2.ca_zip
;

```

| id | operation | E-rows | E-memory | E-width | E-costs |
|-----------|---|------------|----------|---------|------------|
| 1 | -> Row Adapter | 6 | | 273 | 3401632.49 |
| 2 | -> Vector Streaming (type: GATHER) | 6 | | 273 | 3401632.49 |
| 3 | -> Vector Hash Aggregate | 6 | 16MB | 273 | 3401630.82 |
| 4 | -> Vector Streaming(type: REDISTRIBUTE) | 6 | 1MB | 169 | 3401630.78 |
| 5 | -> Vector Hash Join (6,21) | 6 | 16MB | 169 | 3401630.42 |
| 6 | -> Vector Hash Join (7,20) | 7 | 43MB | 173 | 3400343.15 |
| 7 | -> Vector Streaming(type: REDISTRIBUTE) | 7 | 1MB | 123 | 3395775.64 |
| 8 | -> Vector Hash Join (9,19) | 7 | 27MB | 123 | 3395775.48 |
| 9 | -> Vector Streaming(type: REDISTRIBUTE) | 7 | 1MB | 123 | 3386294.72 |
| 10 | -> Vector Hash Join (11,18) | 7 | 16MB | 123 | 3386294.56 |
| 11 | -> Vector Hash Join (12,14) | 7 | 19MB | 112 | 3384018.02 |
| 12 | -> Vector Partition Iterator | 287999764 | 1MB | 12 | 227383.99 |
| 13 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 12 | 227383.99 |
| 14 | -> Vector Hash Join (15,17) | 1516824 | 16MB | 124 | 3065686.08 |
| 15 | -> Vector Partition Iterator | 2879987999 | 1MB | 66 | 2756066.50 |
| 16 | -> Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 66 | 2756066.50 |
| 17 | -> CStore Scan on item | 158 | 1MB | 58 | 4051.25 |
| 18 | -> CStore Scan on store | 24048 | 1MB | 19 | 2264.00 |
| 19 | -> CStore Scan on customer | 12000000 | 1MB | 8 | 12923.00 |
| 20 | -> CStore Scan on customer_address ad2 | 6000000 | 1MB | 58 | 5770.00 |
| 21 | -> CStore Scan on promotion | 36000 | 1MB | 4 | 1268.50 |
| (21 rows) | | | | | |

16.4.9.2 Join Order Hints

Function

These hints specify the join order and outer/inner tables.

Syntax

- Single-layer parentheses (), specifying only the join order. The order of internal and foreign tables is not specified.

```
leading(join_table_list)
```

```
leading(@block_name join_table_list)
```

- Double parentheses (()), specifying the join order and outer/inner tables. The outer/inner tables are specified by the outermost parentheses.

```
leading((join_table_list))
```

```
leading(@block_name (join_table_list))
```

- Single-layer square brackets [], specifying the join order of [] and the sequence of internal and foreign tables.

```
leading[join_table_list]
leading[@block_name join_table_list]
```

- Combination of single-layer parentheses () and single-layer square brackets [], specifying the join order and the sequence of internal and foreign tables at any layer. The parentheses () specify only the join order, but not the sequence of internal and foreign tables. The square brackets [] specify both the join order and the sequence of internal and foreign tables.

```
leading(join_table_list1 [join_table_list2])
leading[join_table_list1 [join_table_list2]]
leading[join_table_list1 (join_table_list2)]
leading(@block_name join_table_list1 [join_table_list2])
leading[@block_name join_table_list1 [join_table_list2]]
leading[@block_name join_table_list1 (join_table_list2)]
```

NOTICE

Single-layer square brackets [] can be used together with single-layer parentheses () to specify the sequence of internal and foreign tables of any layer. Single-layer [] and double-layer () cannot be used together.

Parameter Description

- **join_table_list**

Specifies the tables to be joined. The values can be table names or table aliases. If a subquery is pulled up, the value can also be the subquery alias. Separate the values with spaces. You can add parentheses to specify the join priorities of tables.

To prevent semantic errors, tables in the list must meet the following requirements:

- The tables must exist in the query or its subquery to be pulled up.
- The table names must be unique in the query or subquery to be pulled up. If they are not, their aliases must be unique.
- A table appears only once in the list.
- An alias (if any) is used to represent a table.

NOTE

- The syntax format of the table is as follows:
[schema.]table[@block_name]
The table name can contain the schema name or block name before the subquery statement block is promoted. If the subquery statement block is optimized and rewritten by the optimizer, the value of **block_name** is different from that of **block_name** in leading.
- If a table has an alias, the alias is preferentially used to represent the table.
- **block_name**
Specifies the block name of the statement block. It indicates that the hint takes effect in the subquery statement block corresponding to the block name.

NOTICE

- By default, a block name is generated for a statement.
- CN lightweight statements do not generate block names.
- Block names can be generated for the CREATE TABLE AS SELECT, SELECT INTO, SELECT, INSERT, UPDATE, DELETE, and MERGE statements.
- The naming rule of a block name is as follows:
 - A block name is automatically generated for the SELECT, INSERT, UPDATE, DELETE, and MERGE statements. The naming format of a block name for these statements is sel\$n, ins\$n, upd\$n, del\$n, and mer\$n, respectively, where n starts from 1. The number of statements of different types is not accumulated, but the number of statements of the same type is accumulated.

Example:

```
INSERT INTO t SELECT * FROM t1 WHERE a1 IN (select * from t2);
```

-----sel\$2-----

-----sel\$1-----

-----ins\$1-----

- Recursively assigns a block name to each statement block before the optimizer is used.

First, assign block names to the existing statements block based on the statement type, then traverse the statement blocks in the following sequence, and assign block name to the statement blocks in the statement blocks:

 1. Traverse the target column.
 2. Traverse the target column in the source table of the MERGE statement.
 3. Traverse actions (update or insert) in the MERGE statement.
 4. Traverse the returning clause.
 5. Traverse the Join and Where conditions in From. (The join condition takes precedence over the Where condition.)
 6. For a set operation, traverse each branch of the set (UNION, INTERSECT, and EXCEPT).
 7. Traverse the HAVING clause.
 8. Traverse the LIMIT OFFSET clause.
 9. Traverse the LIMIT COUNT clause.
 10. Traverse CTE
 11. Traverse the table after From.
 12. Traverse the UPSERT clause.
- In the rewriting phase of the optimizer, rewriting optimization is performed due to FULL JOIN, cte inline, materialized view rewriting, INLIST2JOIN, OR conversion, multi count(distinct), Magic Set, lazyagg, and subquery/sublink promotion, a new subquery is constructed. In this case, the recursive processing during block name assignment is also applied to the newly constructed subquery. The number of block names is accumulated.

- In the optimizer rewriting phase, when a subquery is promoted, the table in the inner subquery is promoted to the outer query, and the inner subquery is eliminated. In this case, the promoted table may have the same name as the table in the outer queries. Therefore, the block name to which the promoted table belongs is recorded in the table to distinguish two tables with the same name but are from different query blocks.

For example:

leading(t1 t2 t3 t4 t5): t1, t2, t3, t4, and t5 are joined. The join order and outer/inner tables are not specified.

leading(t1 t2 t3 t4 t5): t1, t2, t3, t4, and t5 are joined in sequence. The table on the right is used as the inner table in each join.

leading(t1 (t2 t3 t4) t5): First, t2, t3, and t4 are joined and the outer/inner tables are not specified. Then, the result is joined with t1 and t5, and the outer/inner tables are not specified.

leading(t1 (t2 t3 t4) t5): First, t2, t3, and t4 are joined and the outer/inner tables are not specified. Then, the result is joined with t1, and (t2 t3 t4) is used as the inner table. Finally, the result is joined with t5, and t5 is used as the inner table.

leading((t1 (t2 t3) t4 t5)) leading((t3 t2)): First, t2 and t3 are joined and t2 is used as the inner table. Then, the result is joined with t1, and (t2 t3) is used as the inner table. Finally, the result is joined with t4 and then t5, and the table on the right in each join is used as the inner table.

leading[t1 [t2 t3]] is equivalent to leading((t1 (t2 t3))) leading((t2 t3)).

leading(t1 [t2 t3]) is equivalent to leading(t1 t2 t3) leading((t2 t3)).

leading[@sel\$1 t1@sel\$1 [t2@sel\$2 t3@sel\$2]] indicates that t2 and t3 are located in the subquery. After the subquery is promoted, t2 and t3 are joined, and then the join table is joined to t1. Where t2 is a foreign table, t3 is an internal table, t1 is a foreign table. The join table of t2 and t3 is an internal table.

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ leading((((store_sales store) promotion) item) customer) ad2) store_returns) leading((store
store_sales))*/ i_product_name product_name ...
```

First, **store_sales** and **store** are joined and **store_sales** is the inner table. Then, The result is joined with **promotion**, **item**, **customer**, **ad2**, and **store_returns** in sequence. The optimized plan is as follows:

WARNING: Duplicated or conflict hint: Leading(store_sales store), will be discarded.

| id | operation | E-rows | E-memory | E-width | E-costs |
|----|---|------------|----------|---------|-------------|
| 1 | -> Row Adapter | 6 | | 273 | 16308094.34 |
| 2 | -> Vector Streaming (type: GATHER) | 6 | | 273 | 16308094.34 |
| 3 | -> Vector Hash Aggregate | 6 | 16MB | 273 | 16308092.67 |
| 4 | -> Vector Hash Join (5,20) | 6 | 585MB | 169 | 16308092.63 |
| 5 | -> Vector Streaming(type: REDISTRIBUTE) | 1320811 | 1MB | 181 | 16069870.93 |
| 6 | -> Vector Hash Join (7,19) | 1320811 | 43MB | 181 | 16061891.00 |
| 7 | -> Vector Streaming(type: REDISTRIBUTE) | 1320811 | 1MB | 131 | 16056566.78 |
| 8 | -> Vector Hash Join (9,18) | 1320811 | 27MB | 131 | 16048586.85 |
| 9 | -> Vector Streaming(type: REDISTRIBUTE) | 1383248 | 1MB | 131 | 16038321.62 |
| 10 | -> Vector Hash Join (11,17) | 1383248 | 16MB | 131 | 16029664.50 |
| 11 | -> Vector Hash Join (12,16) | 2626366951 | 16MB | 73 | 15751384.88 |
| 12 | -> Vector Hash Join (13,14) | 2750085660 | 2156MB | 77 | 14226077.19 |
| 13 | -> CStore Scan on store | 24048 | 1MB | 19 | 2264.00 |
| 14 | -> Vector Partition Iterator | 2879987999 | 1MB | 66 | 2756066.50 |
| 15 | -> Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 66 | 2756066.50 |
| 16 | -> CStore Scan on promotion | 36000 | 1MB | 4 | 1268.50 |
| 17 | -> CStore Scan on item | 158 | 1MB | 58 | 4051.25 |
| 18 | -> CStore Scan on customer | 12000000 | 1MB | 8 | 12923.00 |
| 19 | -> CStore Scan on customer_address ad2 | 6000000 | 1MB | 58 | 5770.00 |
| 20 | -> Vector Partition Iterator | 287999764 | 1MB | 12 | 227383.99 |
| 21 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 12 | 227383.99 |

(21 rows)

For details about the warning at the top of the plan, see [Hint Errors, Conflicts, and Other Warnings](#).

16.4.9.3 Join Operation Hints

Function

Specifies the join method. It can be nested loop join, hash join, or merge join.

Syntax

```
[no] nestloop|hashjoin|mergejoin([@block_name] table_list)
```

Parameter Description

- no** indicates that the specified hint will not be used for a join.
- block_name* indicates the block name of the statement block. For details, see [block_name](#).
- table_list* specifies the tables to be joined. The values are the same as those of [join_table_list](#) but contain no parentheses.

For example:

no nestloop(t1 t2 t3): **nestloop** is not used for joining **t1**, **t2**, and **t3**. The three tables may be joined in either of the two ways: Join **t2** and **t3**, and then **t1**; join **t1** and **t2**, and then **t3**. This hint takes effect only for the last join. If necessary, you can hint other joins. For example, you can add **no nestloop(t2 t3)** to join **t2** and **t3** first and to forbid the use of **nestloop**.

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ nestloop(store_sales store_returns item) */ i_product_name product_name ...
```

nestloop is used for the last join between **store_sales**, **store_returns**, and **item**. The optimized plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|----|---|------------|----------|---------|-----------------|
| 1 | -> Row Adapter | 6 | | 273 | 100061693161.06 |
| 2 | -> Vector Streaming (type: GATHER) | 6 | | 273 | 100061693161.06 |
| 3 | -> Vector Hash Aggregate | 6 | 16MB | 273 | 100061693159.40 |
| 4 | -> Vector Streaming (type: REDISTRIBUTE) | 6 | 1MB | 169 | 100061693159.36 |
| 5 | -> Vector Hash Join (6,22) | 6 | 43MB | 169 | 100061693158.99 |
| 6 | -> Vector Streaming (type: REDISTRIBUTE) | 6 | 1MB | 119 | 100061688591.48 |
| 7 | -> Vector Hash Join (8,21) | 6 | 16MB | 119 | 100061688591.30 |
| 8 | -> Vector Hash Join (9,20) | 7 | 27MB | 123 | 100061687304.04 |
| 9 | -> Vector Streaming (type: REDISTRIBUTE) | 7 | 1MB | 123 | 100061677823.27 |
| 10 | -> Vector Hash Join (11,19) | 7 | 16MB | 123 | 100061677823.12 |
| 11 | -> Vector Nest Loop (12,17) | 7 | 1MB | 112 | 100061675546.57 |
| 12 | -> Vector Hash Join (13,15) | 13670 | 585MB | 62 | 6163443.54 |
| 13 | -> Vector Partition Iterator | 2879987999 | 1MB | 66 | 2756066.50 |
| 14 | -> Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 66 | 2756066.50 |
| 15 | -> Vector Partition Iterator | 287999764 | 1MB | 12 | 227383.99 |
| 16 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 12 | 227383.99 |
| 17 | -> Vector Materialize | 158 | 16MB | 58 | 4051.28 |
| 18 | -> CStore Scan on item | 158 | 1MB | 58 | 4051.25 |
| 19 | -> CStore Scan on store | 24048 | 1MB | 19 | 2264.00 |
| 20 | -> CStore Scan on customer | 12000000 | 1MB | 8 | 12923.00 |
| 21 | -> CStore Scan on promotion | 36000 | 1MB | 4 | 1268.50 |
| 22 | -> CStore Scan on customer_address ad2 | 6000000 | 1MB | 58 | 5770.00 |

(22 rows)

16.4.9.4 Rows Hints

Function

These hints specify the number of rows in an intermediate result set. Both absolute values and relative values are supported.

Syntax

```
rows([@block_name] table_list #|+|-)* const)
```

Parameter Description

- block_name* indicates the block name of the statement block. For details, see [block_name](#).
- #*, *+*, *-*, and *** are operators used for hinting the estimation. *#* indicates that the original estimation is used without any calculation. *+*, *-*, and *** indicate that the original estimation is calculated using these operators. The minimum calculation result is 1. *table_list* indicates the tables to be joined. The values are the same as those of *table_list* in [Join Operation Hints](#).
- const* can be any non-negative number and supports scientific notation.

For example:

rows(t1 #5): The result set of **t1** is five rows.

rows(t1 t2 t3 *1000): Multiply the result set of joined **t1**, **t2**, and **t3** by 1000.

Suggestion

- The hint using *** for two tables is recommended, because this hint will take effect for a join as long as the two tables appear on both sides of this join. For example, if the hint is **rows(t1 t2 * 3)**, the join result of (**t1 t3 t4**) and (**t2 t5 t6**) will be multiplied by 3 because **t1** and **t2** appear on both sides of the join.
- rows** hints can be specified for the result sets of a single table, multiple tables, function tables, and subquery scan tables.

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ rows(store_sales store_returns *50) */ i_product_name product_name ...
```

Multiply the result set of joined **store_sales** and **store_returns** by 50. The optimized plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|-----------|---|------------|----------|---------|------------|
| 1 | -> Row Adapter | 312 | | 273 | 3401656.58 |
| 2 | -> Vector Streaming (type: GATHER) | 312 | | 273 | 3401656.58 |
| 3 | -> Vector Hash Aggregate | 312 | 16MB | 273 | 3401634.91 |
| 4 | -> Vector Streaming(type: REDISTRIBUTE) | 313 | 1MB | 169 | 3401634.39 |
| 5 | -> Vector Hash Join (6,21) | 313 | 43MB | 169 | 3401633.06 |
| 6 | -> Vector Streaming(type: REDISTRIBUTE) | 313 | 1MB | 119 | 3397065.38 |
| 7 | -> Vector Hash Join (8,20) | 313 | 27MB | 119 | 3397064.31 |
| 8 | -> Vector Streaming(type: REDISTRIBUTE) | 328 | 1MB | 119 | 3387583.37 |
| 9 | -> Vector Hash Join (10,19) | 328 | 16MB | 119 | 3387582.18 |
| 10 | -> Vector Hash Join (11,18) | 344 | 16MB | 123 | 3386294.74 |
| 11 | -> Vector Hash Join (12,14) | 360 | 19MB | 112 | 3384018.02 |
| 12 | -> Vector Partition Iterator | 287999764 | 1MB | 12 | 227383.99 |
| 13 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 12 | 227383.99 |
| 14 | -> Vector Hash Join (15,17) | 1516824 | 16MB | 124 | 3065686.08 |
| 15 | -> Vector Partition Iterator | 2879987999 | 1MB | 66 | 2756066.50 |
| 16 | -> Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 66 | 2756066.50 |
| 17 | -> CStore Scan on item | 158 | 1MB | 58 | 4051.25 |
| 18 | -> CStore Scan on store | 24048 | 1MB | 19 | 2264.00 |
| 19 | -> CStore Scan on promotion | 36000 | 1MB | 4 | 1268.50 |
| 20 | -> CStore Scan on customer | 12000000 | 1MB | 8 | 12923.00 |
| 21 | -> CStore Scan on customer_address ad2 | 6000000 | 1MB | 58 | 5770.00 |
| (21 rows) | | | | | |

The estimation value after the hint in row 11 is **360**, and the original value is rounded off to 7.

16.4.9.5 Stream Operation Hints

Function

Specifies the stream method, which can be broadcast, redistribute, or specifying the distribution key for **Agg** redistribution.

NOTE

Specifies the hint for the distribution column during the Agg process. This parameter is supported only by clusters of version 8.1.3.100 or later.

Syntax

```
[no] broadcast | redistribute([@block_name] table_list) | redistribute ([@block_name] (*) (columns))
```

Parameter Description

- **no** indicates that the hinted stream method is not used. When the hint is specified for the distribution columns in the **Agg** redistribution, **no** is invalid.
- *block_name* indicates the block name of the statement block. For details, see [block_name](#).
- *table_list* specifies the tables to be joined. For details, see [Parameter Description](#).
- When hints are specified for distribution columns, the asterisk (*) is fixed and the table name cannot be specified.
- **columns** specifies one or more columns in the **GROUP BY** clause. When there are no **GROUP BY** clauses, it can specify the columns in the **DISTINCT** clause.

NOTE

- The specified distribution column must be specified using the column sequence number or column name in **group by** or **distinct**. The columns in **count(distinct)** can only be specified using column names.
- For a multi-layer query, you can specify the distribution column hint at each layer. The hint takes effect only at the corresponding layer.
- The column specified in **count(distinct)** takes effect only for two-level hashagg plans. Otherwise, the specified distribution column is invalid.
- If the optimizer finds that redistribution is not required after estimation, the specified distribution column is invalid.

Tips

- Generally, the optimizer selects a group of non-skew distribution keys for data redistribution based on statistics. If the default distribution keys have data skew, you can manually specify the distribution columns to avoid data skew.
- When selecting a distribution key, select a group of columns with high distinct values as the distribution key based on data distribution features. In this way, data can be evenly distributed to each DN after redistribution.
- After writing hints, you can run **explain verbose** to print the execution plan and check whether the specified distribution key is valid. If the specified distribution key is invalid, a warning is displayed.

Example

- Hint the query plan in [Examples](#) as follows:

```
explain
```

```
select /*+ no redistribute(store_sales store_returns item store) leading(((store_sales store_returns item store) customer)) */ i_product_name product_name ...
```

In the original plan, the join result of **store_sales**, **store_returns**, **item**, and **store** is redistributed before it is joined with **customer**. After the hinting, the redistribution is disabled and the join order is retained. The optimized plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|----|---|------------|----------|---------|------------|
| 1 | -> Row Adapter | 6 | | 273 | 5718448.94 |
| 2 | -> Vector Streaming (type: GATHER) | 6 | | 273 | 5718448.94 |
| 3 | -> Vector Hash Aggregate | 6 | 16MB | 273 | 5718447.27 |
| 4 | -> Vector Streaming(type: REDISTRIBUTE) | 6 | 1MB | 169 | 5718447.23 |
| 5 | -> Vector Hash Join (6,21) | 6 | 16MB | 169 | 5718446.86 |
| 6 | -> Vector Hash Join (7,20) | 7 | 43MB | 173 | 5717159.60 |
| 7 | -> Vector Streaming(type: REDISTRIBUTE) | 7 | 1MB | 123 | 5712592.09 |
| 8 | -> Vector Hash Join (9,18) | 7 | 585MB | 123 | 5712591.93 |
| 9 | -> Vector Hash Join (10,17) | 7 | 16MB | 123 | 3386294.56 |
| 10 | -> Vector Hash Join (11,13) | 7 | 19MB | 112 | 3384018.02 |
| 11 | -> Vector Partition Iterator | 287999764 | 1MB | 12 | 227383.99 |
| 12 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 12 | 227383.99 |
| 13 | -> Vector Hash Join (14,16) | 1516824 | 16MB | 124 | 3065686.08 |
| 14 | -> Vector Partition Iterator | 2879987999 | 1MB | 66 | 2756066.50 |
| 15 | -> Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 66 | 2756066.50 |
| 16 | -> CStore Scan on item | 158 | 1MB | 58 | 4051.25 |
| 17 | -> CStore Scan on store | 24048 | 1MB | 19 | 2264.00 |
| 18 | -> Vector Streaming(type: BROADCAST) | 288000000 | 1MB | 8 | 2176297.36 |
| 19 | -> CStore Scan on customer | 12000000 | 1MB | 8 | 12923.00 |
| 20 | -> CStore Scan on customer_address ad2 | 6000000 | 1MB | 58 | 5770.00 |
| 21 | -> CStore Scan on promotion | 36000 | 1MB | 4 | 1268.50 |

(21 rows)

- Specifies the distribution columns for Agg redistribution.

```
explain (verbose on, costs off, nodes off)
```

```
select /*+ redistribute ((*) (2 3)) */ a1, b1, c1, count(c1) from t1 group by a1, b1, c1 having count(c1) > 10 and sum(d1) > 100
```

In the following example, the last two columns of the specified **GROUP BY** columns are used as distribution keys.

```

                                QUERY PLAN
-----
id | operation
---+-----
1 | -> Streaming (type: GATHER)
2 |   -> HashAggregate
3 |     -> Streaming(type: REDISTRIBUTE)
4 |       -> Seq Scan on public.t1

      Predicate Information (identified by plan id)
-----
2 --HashAggregate
   Filter: ((count(t1.c1) > 10) AND (sum(t1.d1) > 100))

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
   Output: a1, b1, c1, (count(c1))
2 --HashAggregate
   Output: a1, b1, c1, count(c1)
   Group By Key: t1.a1, t1.b1, t1.c1
3 --Streaming(type: REDISTRIBUTE)
   Output: a1, b1, c1, d1
   Distribute Key: b1, c1
4 --Seq Scan on public.t1
   Output: a1, b1, c1, d1

===== Query Summary =====
-----
System available mem: 24862720KB
Query Max mem: 24862720KB
Query estimated mem: 3138KB
(30 rows)
```

- If the statement does not contain the **GROUP BY** clause, specify the distinct column as the distribution columns.

explain (verbose on, costs off, nodes off)

```
select /*+ redistribute ((* (3 1)) */ distinct a1, b1, c1 from t1;
```

```

                                QUERY PLAN
-----
id | operation
---+-----
1 | -> Streaming (type: GATHER)
2 |   -> HashAggregate
3 |     -> Streaming(type: REDISTRIBUTE)
4 |       -> Seq Scan on public.t1

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
   Output: a1, b1, c1
2 --HashAggregate
   Output: a1, b1, c1
   Group By Key: t1.a1, t1.b1, t1.c1
3 --Streaming(type: REDISTRIBUTE)
   Output: a1, b1, c1
   Distribute Key: c1, a1
4 --Seq Scan on public.t1
   Output: a1, b1, c1

===== Query Summary =====
-----
System available mem: 24862720KB
Query Max mem: 24862720KB
Query estimated mem: 3136KB
(25 rows)
```

16.4.9.6 Scan Operation Hints

Function

These hints specify a scan operation, which can be **tablescan**, **indexscan**, or **indexonlyscan**.

Syntax

```
[no] tablescan|indexscan|indexonlyscan([@block_name] table [index])
```

Parameter Description

- **no** indicates that the specified hint will not be used for a join.
- *block_name* indicates the block name of the statement block. For details, see [block_name](#).
- *table* specifies the table to be scanned. You can specify only one table. Use a table alias (if any) instead of a table name.

NOTE

- The syntax format of the table is as follows:
[schema.]table[@block_name]
The table name can contain the schema name or block name before the subquery statement block is promoted. If the subquery statement block is optimized and rewritten by the optimizer, the value of **block_name** is different from that of **block_name** in leading.
- If a table has an alias, the alias is preferentially used to represent the table.
- index* indicates the index for **indexscan** or **indexonlyscan**. You can specify only one index.

NOTE

indexscan and **indexonlyscan** hints can be used only when the specified index belongs to the table.

Scan operation hints can be used for row-store tables, column-store tables, HDFS tables, HDFS foreign tables, OBS tables, and subquery tables. HDFS tables include primary tables and delta tables. The delta tables are invisible to users. Therefore, scan operation hints are used only for primary tables.

If **indexscan** is specified, **indexscan** or **indexonlyscan** takes effect. **indexscan** and **indexonlyscan** can also take effect at the same time. When **indexscan** and **indexonlyscan** hints appear at the same time, **indexonlyscan** takes effect first.

Example

To specify an index-based hint for a scan, create an index named **i** on the **i_item_sk** column of the **item** table.

```
create index i on item(i_item_sk);
```

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ indexscan(item i) */ i_product_name product_name ...
```

item is scanned based on an index. The optimized plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|-----------|---|------------|----------|---------|-----------------|
| 1 | -> Row Adapter | 6 | | 273 | 100061674938.26 |
| 2 | -> Vector Streaming (type: GATHER) | 6 | | 273 | 100061674938.26 |
| 3 | -> Vector Hash Aggregate | 6 | 16MB | 273 | 100061674936.59 |
| 4 | -> Vector Streaming (type: REDISTRIBUTE) | 6 | 1MB | 169 | 100061674936.55 |
| 5 | -> Vector Hash Join (6,21) | 6 | 43MB | 169 | 100061674936.19 |
| 6 | -> Vector Streaming (type: REDISTRIBUTE) | 6 | 1MB | 119 | 100061670368.67 |
| 7 | -> Vector Hash Join (8,20) | 6 | 16MB | 119 | 100061670368.50 |
| 8 | -> Vector Hash Join (9,19) | 7 | 27MB | 123 | 100061669081.23 |
| 9 | -> Vector Streaming (type: REDISTRIBUTE) | 7 | 1MB | 123 | 100061659600.47 |
| 10 | -> Vector Hash Join (11,18) | 7 | 16MB | 123 | 100061659600.31 |
| 11 | -> Vector Nest Loop (12,17) | 7 | 1MB | 112 | 100061657323.77 |
| 12 | -> Vector Hash Join (13,15) | 13670 | 585MB | 62 | 6163443.54 |
| 13 | -> Vector Partition Iterator | 2879987999 | 1MB | 66 | 2756066.50 |
| 14 | -> Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 66 | 2756066.50 |
| 15 | -> Vector Partition Iterator | 287999764 | 1MB | 12 | 227383.99 |
| 16 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 12 | 227383.99 |
| 17 | -> CStore Index Scan using i on item | 1 | 1MB | 58 | 4.01 |
| 18 | -> CStore Scan on store | 24048 | 1MB | 19 | 2264.00 |
| 19 | -> CStore Scan on customer | 12000000 | 1MB | 8 | 12923.00 |
| 20 | -> CStore Scan on promotion | 36000 | 1MB | 4 | 1268.50 |
| 21 | -> CStore Scan on customer_address ad2 | 6000000 | 1MB | 58 | 5770.00 |
| (21 rows) | | | | | |

16.4.9.7 Sublink Name Hints

Function

These hints specify the name of a sublink block.

Syntax

```
blockname ([@block_name] table)
```

Precautions

- This block name hint is used by an outer query only when a sublink is pulled up. Currently, only the **Agg** equivalent join, **IN**, and **EXISTS** sublinks can be pulled up. This hint is usually used together with the hints described in the previous sections.
- The subquery after the **FROM** keyword is hinted by using the subquery alias. In this case, **block_name hint** becomes invalid.
- If a sublink contains multiple tables, the tables will be joined with the outer-query tables in a random sequence after the sublink is pulled up. In this case, **blockname** also becomes invalid.

Parameter Description

- *block_name* indicates the block name of the statement block. For details, see [block_name](#).
- *table* indicates the name you have specified for a sublink block.

NOTE

- The syntax format of the table is as follows:
[schema.]table[@block_name]
The table name can contain the schema name or block name before the subquery statement block is promoted. If the subquery statement block is optimized and rewritten by the optimizer, the value of **block_name** is different from that of **block_name** in leading.
- If a table has an alias, the alias is preferentially used to represent the table.

Example

```
explain select /*+nestloop(store_sales tt) */ * from store_sales where ss_item_sk in (select /*
+blockname(tt)*/ i_item_sk from item group by 1);
```

tt indicates the sublink block name. After being pulled up, the sublink is joined with the outer-query table **store_sales** by using **nestloop**. The optimized plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|----------|---|------------|----------|---------|-----------------|
| 1 | -> Row Adapter | 1439994000 | | 216 | 325105765847.91 |
| 2 | -> Vector Streaming (type: GATHER) | 1439994000 | | 216 | 325105765847.91 |
| 3 | -> Vector Nest Loop Semi Join (4, 6) | 1439994000 | 1MB | 216 | 325026664615.00 |
| 4 | -> Vector Partition Iterator | 2879987999 | 1MB | 216 | 2756066.50 |
| 5 | -> Partitioned CStore Scan on store_sales | 2879987999 | 1MB | 216 | 2756066.50 |
| 6 | -> Vector Materialize | 300000 | 16MB | 4 | 4176.25 |
| 7 | -> Vector Hash Aggregate | 300000 | 16MB | 4 | 3988.75 |
| 8 | -> CStore Scan on item | 300000 | 1MB | 4 | 3832.50 |
| (8 rows) | | | | | |

16.4.9.8 Skew Hints

Function

These hints specify redistribution keys containing skew data and skew values, and are used to optimize redistribution involving Join or HashAgg.

Precautions

- Skew hints are used only if redistribution is required and the specified skew information matches the redistribution information.
- Skew hints are controlled by the GUC parameter [skew_option](#). If the parameter is disabled, skew hints cannot be used for solving skew.
- Currently, skew hints support only the table relationships of the ordinary table and subquery types. Hints can be specified for base tables, subqueries, and **WITH ... AS** clauses. Unlike other hints, a subquery can be used in skew hints regardless of whether it is pulled up.
- Use an alias (if any) to specify a table where data skew occurs.
- You can use a name or an alias to specify a skew column as long as it is not ambiguous. The columns in skew hints cannot be expressions. If data skew occurs in the redistribution that uses an expression as a redistribution key, set the redistribution key as a new column and specify the column in skew hints.
- The number of skew values must be an integer multiple of the number of columns. Skew values must be grouped based on the column sequence, with each group containing a maximum of 10 values. You can specify duplicate values to group skew columns having different number of skew values. For example, the **c1** and **c2** columns of the **t1** table contains skew data. The skew value of the **c1** column is **a1**, and the skew values of the **c2** column are **b1** and **b2**. In this case, the skew hint is **skew(t1 (c1 c2) ((a1 b1)(a1 b2)))**. (**a1 b1**) is a value group, where **NULL** is allowed as a skew value. Each hint can contain a maximum of 10 groups and the number of groups should be an integer multiple of the number of columns.
- In the redistribution optimization of Join, a skew value must be specified for skew hints. The skew value can be left empty for HashAgg.
- If multiple tables, columns, or values are specified, separate items of the same type with spaces.
- The type of skew values cannot be forcibly converted in hints. To specify a string, enclose it with single quotation marks (' ').

Syntax

- Specify single-table skew.
`skew([@block_name] table (column) [(value)])`
- Specify intermediate result skew.
`skew([@block_name] (join_rel) (column) [(value)])`

Parameter Description

- *block_name* indicates the block name of the statement block. For details, see [block_name](#).
- **table** specifies the table where skew occurs.

 NOTE

- The syntax format of the table is as follows:
[schema.]table[@block_name]
The table name can contain the schema name or block name before the subquery statement block is promoted. If the subquery statement block is optimized and rewritten by the optimizer, the value of **block_name** is different from that of **block_name** in leading.
- If a table has an alias, the alias is preferentially used to represent the table.
- **join_rel** specifies two or more joined tables. For example, **(t1 t2)** indicates that the result of joining **t1** and **t2** tables contains skew data.
- **column** specifies one or more columns where skew occurs.
- **value** specifies one or more skew values.

Example:

- Specify single-table skew.
Each skew hint describes the skew information of one table relationship. To describe the skews of multiple table relationships in a query, specify multiple skew hints.
Skew hints have the following formats:
 - One skew value in one column: **skew(t (c1) (v1))**
Description: The **v1** value in the **c1** column of the **t** table relationship causes skew in query execution.
 - Multiple skew values in one column: **skew(t (c1) (v1 v2 v3 ...))**
Description: Values including **v1**, **v2**, and **v3** in the **c1** column of the **t** table relationship cause skew in query execution.
 - Multiple columns, each having one skew value: **skew(t (c1 c2) (v1 v2))**
Description: The **v1** value in the **c1** column and the **v2** value in the **c2** column of the **t** table relationship cause skew in query execution.
 - Multiple columns, each having multiple skew values: **skew(t (c1 c2) ((v1 v2) (v3 v4) (v5 v6) ...))**
Description: Values including **v1**, **v3**, and **v5** in the **c1** column and values including **v2**, **v4**, and **v6** in the **c2** column of the **t** table relationship cause skew in query execution.

NOTICE

In the last format, parentheses for skew value groups can be omitted, for example, **skew(t (c1 c2) (v1 v2 v3 v4 v5 v6 ...))**. In a skew hint, either use parentheses for all skew value groups or for none of them.

Otherwise, a syntax error will be generated. For example, **skew(t (c1 c2) (v1 v2 v3 v4 (v5 v6) ...))** will generate an error.

- Specify intermediate result skew.
If data skew does not occur in base tables but in an intermediate result during query execution, specify skew hints of the intermediate result to solve the skew. The format is **skew((t1 t2) (c1) (v1))**.

Description: Data skew occurs after the table relationships **t1** and **t2** are joined. The **c1** column of the **t1** table contains skew data and its skew value is **v1**.

c1 can exist only in a table relationship of **join_rel**. If there is another column having the same name, use aliases to avoid ambiguity.

Suggestion

- For a multi-level query, write the hint on the layer where data skew occurs.
- For a listed subquery, you can specify the subquery name in a hint. If you know data skew occurs on which base table, directly specify the table.
- Aliases are preferred when you specify a table or column in a hint.

Examples

Specify single-table skew.

- Specify hints in the original query.
For example, the original query is as follows:

```
explain
with customer_total_return as
(select sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
select  c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

| id | operation | E-rows | E-memory | E-width | E-costs |
|-----------|---|-----------|-----------------|---------|-----------|
| 1 | -> Row Adapter | 100 | | 20 | 911254.47 |
| 2 | -> Vector Limit | 100 | | 20 | 911254.47 |
| 3 | -> Vector Streaming (type: GATHER) | 2400 | | 20 | 911325.75 |
| 4 | -> Vector Limit | 2400 | 1MB | 20 | 911247.62 |
| 5 | -> Vector Sort | 3684816 | 16MB | 20 | 911631.21 |
| 6 | -> Vector Hash Join (7,29) | 3684817 | 41MB (12374MB) | 20 | 905379.41 |
| 7 | -> Vector Streaming (type: REDISTRIBUTE) | 3684817 | 384KB | 4 | 883010.31 |
| 8 | -> Vector Hash Join (9,19) | 3684817 | 16MB | 4 | 861302.05 |
| 9 | -> Vector Hash Join (10,18) | 11054450 | 16MB | 44 | 427109.71 |
| 10 | -> Vector Hash Aggregate | 50247501 | 397MB (12671MB) | 54 | 395302.57 |
| 11 | -> Vector Streaming (type: REDISTRIBUTE) | 50247501 | 384KB | 22 | 358663.76 |
| 12 | -> Vector Hash Join (13,15) | 50247501 | 16MB | 22 | 294300.51 |
| 13 | -> Vector Partition Iterator | 287999764 | 1MB | 26 | 227383.99 |
| 14 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 26 | 227383.99 |
| 15 | -> Vector Streaming (type: BROADCAST) | 8712 | 384KB | 4 | 975.56 |
| 16 | -> Vector Partition Iterator | 363 | 1MB | 4 | 910.65 |
| 17 | -> Partitioned CStore Scan on date_dim | 363 | 1MB | 4 | 910.65 |
| 18 | -> CStore Scan on store | 44 | 1MB | 4 | 1006.39 |
| 19 | -> Vector Hash Aggregate | 192 | 16MB | 68 | 426707.38 |
| 20 | -> Vector Subquery Scan on ctr2 | 50247501 | 1MB | 36 | 416239.03 |
| 21 | -> Vector Hash Aggregate | 50247501 | 397MB (12671MB) | 54 | 395302.57 |
| 22 | -> Vector Streaming (type: REDISTRIBUTE) | 50247501 | 384KB | 22 | 358663.76 |
| 23 | -> Vector Hash Join (24,26) | 50247501 | 16MB | 22 | 294300.51 |
| 24 | -> Vector Partition Iterator | 287999764 | 1MB | 26 | 227383.99 |
| 25 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 26 | 227383.99 |
| 26 | -> Vector Streaming (type: BROADCAST) | 8712 | 384KB | 4 | 975.56 |
| 27 | -> Vector Partition Iterator | 363 | 1MB | 4 | 910.65 |
| 28 | -> Partitioned CStore Scan on date_dim | 363 | 1MB | 4 | 910.65 |
| 29 | -> CStore Scan on customer | 12000000 | 1MB | 24 | 12923.00 |
| (29 rows) | | | | | |

Specify the hints of HashAgg in the inner **with** clause and of the outer Hash Join. The query containing hints is as follows:


```
explain
with customer_total_return as
(select /*+ skew(store_returns(sr_store_sk sr_customer_sk)) */sr_customer_sk as ctr_customer_sk
,sr_store_sk as ctr_store_sk
,sum(SR_FEE) as ctr_total_return
from store_returns
,date_dim
where sr_returned_date_sk = d_date_sk
and d_year =2000
group by sr_customer_sk
,sr_store_sk)
select /*+ skew(ctr1(ctr_customer_sk)(11))*/ c_customer_id
from customer_total_return ctr1
,store
,customer
where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
from customer_total_return ctr2
where ctr1.ctr_store_sk = ctr2.ctr_store_sk)
and s_store_sk = ctr1.ctr_store_sk
and s_state = 'NM'
and ctr1.ctr_customer_sk = c_customer_sk
order by c_customer_id
limit 100;
```

The hints indicate that the **group by** in the inner **with** clause contains skew data during redistribution by HashAgg, corresponding to the original Hash Agg operators 10 and 21; and that the **ctr_customer_sk** column in the outer **ctr1** table contains skew data during redistribution by Hash Join, corresponding to operator 6 in the original plan. The optimized plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|----|---|-----------|-----------------|---------|------------|
| 1 | -> Row Adapter | 100 | | 20 | 1061778.14 |
| 2 | -> Vector Limit | 100 | | 20 | 1061778.14 |
| 3 | -> Vector Streaming (type: GATHER) | 2400 | | 20 | 1061849.41 |
| 4 | -> Vector Limit | 2400 | 1MB | 20 | 1061771.29 |
| 5 | -> Vector Sort | 3684816 | 16MB | 20 | 1062154.87 |
| 6 | -> Vector Hash Join (7,31) | 3684817 | 41MB (12344MB) | 20 | 1055903.08 |
| 7 | -> Vector Streaming (type: PART REDISTRIBUTE PART ROUNDROBIN) | 3684817 | 384KB | 4 | 1013056.49 |
| 8 | -> Vector Hash Join (9,20) | 3684817 | 16MB | 4 | 1000006.10 |
| 9 | -> Vector Hash Join (10,19) | 11054450 | 16MB | 44 | 496461.73 |
| 10 | -> Vector Hash Aggregate | 50247501 | 397MB (12010MB) | 54 | 464654.59 |
| 11 | -> Vector Streaming (type: REDISTRIBUTE) | 50247501 | 384KB | 54 | 428015.79 |
| 12 | -> Vector Hash Aggregate | 50247501 | 397MB (12010MB) | 54 | 330939.31 |
| 13 | -> Vector Hash Join (14,16) | 50247501 | 16MB | 22 | 294300.51 |
| 14 | -> Vector Partition Iterator | 287999764 | 1MB | 26 | 227383.99 |
| 15 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 26 | 227383.99 |
| 16 | -> Vector Streaming (type: BROADCAST) | 8712 | 384KB | 4 | 975.56 |
| 17 | -> Vector Partition Iterator | 363 | 1MB | 4 | 910.65 |
| 18 | -> Partitioned CStore Scan on date_dim | 363 | 1MB | 4 | 910.65 |
| 19 | -> CStore Scan on store | 44 | 1MB | 4 | 1006.39 |
| 20 | -> Vector Hash Aggregate | 192 | 16MB | 68 | 496059.40 |
| 21 | -> Vector Subquery Scan on ctr2 | 50247501 | 1MB | 36 | 485591.05 |
| 22 | -> Vector Hash Aggregate | 50247501 | 397MB (12010MB) | 54 | 464654.59 |
| 23 | -> Vector Streaming (type: REDISTRIBUTE) | 50247501 | 384KB | 54 | 428015.79 |
| 24 | -> Vector Hash Aggregate | 50247501 | 397MB (12010MB) | 54 | 330939.31 |
| 25 | -> Vector Hash Join (26,28) | 50247501 | 16MB | 22 | 294300.51 |
| 26 | -> Vector Partition Iterator | 287999764 | 1MB | 26 | 227383.99 |
| 27 | -> Partitioned CStore Scan on store_returns | 287999764 | 1MB | 26 | 227383.99 |
| 28 | -> Vector Streaming (type: BROADCAST) | 8712 | 384KB | 4 | 975.56 |
| 29 | -> Vector Partition Iterator | 363 | 1MB | 4 | 910.65 |
| 30 | -> Partitioned CStore Scan on date_dim | 363 | 1MB | 4 | 910.65 |
| 31 | -> Vector Streaming (type: PART LOCAL PART BROADCAST) | 12000000 | 384KB | 24 | 38485.50 |
| 32 | -> CStore Scan on customer | 12000000 | 1MB | 24 | 12923.00 |

(32 rows)

To solve data skew in the redistribution, Hash Agg is changed to double-level Agg operators and the redistribution operators used by Hash Join are changed in the optimized plan.

- Modify the query and then specify hints.

For example, the original query and its plan are as follows:

```
explain select count(*) from store_sales_1 group by round(ss_list_price);
```

| id | operation | E-rows | E-memory | E-width | E-costs |
|----|---|---------|----------|---------|----------|
| 1 | -> Row Adapter | 16672 | | 14 | 62261.28 |
| 2 | -> Vector Streaming (type: GATHER) | 16672 | | 14 | 62261.28 |
| 3 | -> Vector Streaming (type: LOCAL GATHER dop: 1/2) | 16672 | 32KB | 14 | 61479.78 |
| 4 | -> Vector Hash Aggregate | 16672 | 16MB | 14 | 61452.00 |
| 5 | -> Vector Streaming (type: SPLIT REDISTRIBUTE dop: 2/2) | 3112836 | 128KB | 6 | 57498.43 |
| 6 | -> CStore Scan on store_sales_1 | 3112836 | 1MB | 6 | 21810.25 |

(6 rows)

Columns in hints do not support expressions. To specify hints, rewrite the query as several subqueries. The rewritten query and its plan are as follows:

```
explain
select count(*)
```

```
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

| 1 | | -> | Row Adapter | | 16672 | | | | 14 62261.28 |
|----------|--|----|--|--|---------|--|-------|--|---------------|
| 2 | | -> | Vector Streaming (type: GATHER) | | 16672 | | | | 14 62261.28 |
| 3 | | -> | Vector Streaming (type: LOCAL GATHER dop: 1/2) | | 16672 | | 32KB | | 14 61479.78 |
| 4 | | -> | Vector Hash Aggregate | | 16672 | | 16MB | | 14 61452.00 |
| 5 | | -> | Vector Streaming (type: SPLIT REDISTRIBUTE dop: 2/2) | | 3112836 | | 128KB | | 6 57498.43 |
| 6 | | -> | CStore Scan on store_sales_1 | | 3112836 | | 1MB | | 6 21810.25 |
| (6 rows) | | | | | | | | | |

Ensure that the service logic is not changed during the rewriting.

Specify hints in the rewritten query as follows:

```
explain
select /*+ skew(tmp(a)) */ count(*)
from (select round(ss_list_price),ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

| id | | operation | E-rows | E-memory | E-width | E-costs | |
|----------|--|-----------|--|----------|---------|---------|-----------------------|
| 1 | | -> | Row Adapter | | 16672 | | 14 27771.82 |
| 2 | | -> | Vector Streaming (type: GATHER) | | 16672 | | 14 27771.82 |
| 3 | | -> | Vector Streaming (type: LOCAL GATHER dop: 1/2) | | 16672 | | 32KB 14 26990.32 |
| 4 | | -> | Vector Hash Aggregate | | 16671 | | 16MB 14 26962.54 |
| 5 | | -> | Vector Streaming (type: SPLIT REDISTRIBUTE dop: 2/2) | | 66216 | | 128KB 14 26838.09 |
| 6 | | -> | Vector Hash Aggregate | | 66216 | | 16MB 14 25949.61 |
| 7 | | -> | CStore Scan on store_sales_1 | | 3112836 | | 1MB 6 21810.25 |
| (7 rows) | | | | | | | |

The plan shows that after Hash Agg is changed to double-layer Agg operators, redistributed data is greatly reduced and redistribution time shortened.

You can specify hints in columns in a subquery, for example:

```
explain
select /*+ skew(tmp(b)) */ count(*)
from (select round(ss_list_price) b,ss_hdemo_sk
from store_sales_1)tmp(a,ss_hdemo_sk)
group by a;
```

16.4.9.9 Hint That Disables Subquery Pull-up

Function

To optimize query logic, the optimizer usually pulls up subqueries for execution. However, sometimes the pulled up subqueries do not run much faster than others, and may even be slower due to enlarged search scope. In this case, you can specify the **no merge** hint to disable pull-up. This hint is not recommended in most cases.

Syntax

```
no merge[@block_name]
no merge ([@block_name1] subquery_name[@block_name2])
```

Description

- block_name** indicates the block name of the statement block. For details, see [block_name](#).
- subquery_name** indicates the name of a subquery. It can also be a view or CTE name. The specified subquery will not be unnested during logic optimization. If **subquery_name** is not specified, the current query will not be unnested.

Example

Create tables **t1**, **t2**, and **t3**.

```
create table t1(a1 int,b1 int,c1 int,d1 int);
create table t2(a2 int,b2 int,c2 int,d2 int);
create table t3(a3 int,b3 int,c3 int,d3 int);
```

The original statement is as follows:

```
explain select * from t3, (select a1,b2,c1,d2 from t1,t2 where t1.a1=t2.a2) s1 where t3.b3=s1.b2;
```

| id | operation | E-rows | E-width | E-costs |
|----|--------------------|--------|---------|---------|
| 1 | -> Hash Join (2,6) | 44450 | 32 | 754.31 |
| 2 | -> Hash Join (3,4) | 8885 | 28 | 182.11 |
| 3 | -> Seq Scan on t3 | 1776 | 16 | 27.76 |
| 4 | -> Hash | 1776 | 12 | 27.76 |
| 5 | -> Seq Scan on t2 | 1776 | 12 | 27.76 |
| 6 | -> Hash | 1776 | 8 | 27.76 |
| 7 | -> Seq Scan on t1 | 1776 | 8 | 27.76 |

In this query, you can use the following methods to disable the pull-up of subquery **s1**:

- Method 1:
explain select /*+ no merge(s1) */ * from t3, (select a1,b2,c1,d2 from t1,t2 where t1.a1=t2.a2) s1 where t3.b3=s1.b2;
- Method 2:
explain select * from t3, (select /*+ no merge */ a1,b2,c1,d2 from t1,t2 where t1.a1=t2.a2) s1 where t3.b3=s1.b2;

Outcome:

| id | operation | E-rows | E-width | E-costs |
|----|--------------------|--------|---------|---------|
| 1 | -> Hash Join (2,6) | 8880 | 32 | 443.03 |
| 2 | -> Hash Join (3,4) | 8885 | 16 | 182.11 |
| 3 | -> Seq Scan on t1 | 1776 | 8 | 27.76 |
| 4 | -> Hash | 1776 | 12 | 27.76 |
| 5 | -> Seq Scan on t2 | 1776 | 12 | 27.76 |
| 6 | -> Hash | 1776 | 16 | 27.76 |
| 7 | -> Seq Scan on t3 | 1776 | 16 | 27.76 |

16.4.9.10 Drive Hints

Function

When generating a query plan, the optimizer uses the dynamic planning or genetic algorithm to enumerate possible join paths and selects the optimal path. When the number of join tables increases, the search space may expand greatly. In this case, you can use a drive hint to specify the fact table in the query and use heuristic search to narrow the search range. This hint takes effect only when the GUC parameter `join_search_mode` is set to **heuristic**.

Syntax

```
[no] drive (table)
```

Description

- **no** indicates that the table specified by the hint is not a drive table.
- *table* specifies the table specified by the hint. You can specify only one table. Use a table alias (if any) instead of a table name.

Example

Create tables **t1**, **t2**, and **t3**.

```
create table t1(a1 int,b1 int,c1 int,d1 int);
create table t2(a2 int,b2 int,c2 int,d2 int);
create table t3(a3 int,b3 int,c3 int,d3 int);
```

The original statement is as follows:

```
explain select * from t1,t2,t3 where t1.b1=t2.b2 and t1.c1=t3.c3 and t2.d2=t3.d3;
```

| QUERY PLAN | | | | | |
|------------|-------------------------------|--------|----------|---------|---------|
| id | operation | E-rows | E-memory | E-width | E-costs |
| 1 | -> Streaming (type: GATHER) | 4 | | 48 | 40.03 |
| 2 | -> Hash Join (3,9) | 4 | 1MB | 48 | 34.03 |
| 3 | -> Streaming(type: BROADCAST) | 40 | 2MB | 32 | 23.64 |
| 4 | -> Hash Join (5,7) | 20 | 1MB | 32 | 22.06 |
| 5 | -> Streaming(type: BROADCAST) | 40 | 2MB | 16 | 11.67 |
| 6 | -> Seq Scan on t1 | 20 | 1MB | 16 | 10.10 |
| 7 | -> Hash | 20 | 16MB | 16 | 10.10 |
| 8 | -> Seq Scan on t2 | 20 | 1MB | 16 | 10.10 |
| 9 | -> Hash | 20 | 16MB | 16 | 10.10 |
| 10 | -> Seq Scan on t3 | 20 | 1MB | 16 | 10.10 |

In the preceding query, you can specify table **t3** as the drive table so that **t3** is preferentially joined with other tables. This reduces the search paths generated by the plan and changes the join sequence. :

```
explain select /*+ drive(t3) */ * from t1, t2, t3 where t1.b1=t2.b2 and t1.c1=t3.c3 and t2.d2=t3.d3;
```

After hints are used:

| QUERY PLAN | | | | | |
|------------|-------------------------------|--------|----------|---------|---------|
| id | operation | E-rows | E-memory | E-width | E-costs |
| 1 | -> Streaming (type: GATHER) | 4 | | 48 | 40.03 |
| 2 | -> Hash Join (3,9) | 4 | 1MB | 48 | 34.03 |
| 3 | -> Streaming(type: BROADCAST) | 40 | 2MB | 32 | 23.64 |
| 4 | -> Hash Join (5,7) | 20 | 1MB | 32 | 22.06 |
| 5 | -> Streaming(type: BROADCAST) | 40 | 2MB | 16 | 11.67 |
| 6 | -> Seq Scan on t3(DRIVE) | 20 | 1MB | 16 | 10.10 |
| 7 | -> Hash | 20 | 16MB | 16 | 10.10 |
| 8 | -> Seq Scan on t1 | 20 | 1MB | 16 | 10.10 |
| 9 | -> Hash | 20 | 16MB | 16 | 10.10 |
| 10 | -> Seq Scan on t2 | 20 | 1MB | 16 | 10.10 |

CAUTION

If the GUC parameter **join_search_mode** is set to **heuristic** and the number of joined tables exceeds the value of **from_collapse_limit**, the optimizer automatically identifies drive tables and displays them in the plan. If the final drive selected by the optimizer is incorrect, you can use **no drive hint** to correct the selection.

16.4.9.11 Dictionary Code Hint

Function

Specifies a column to construct a dictionary code. The comparison of character strings in the dictionary code is converted into the comparison of numbers, which accelerates the query speed such as **group by** and **filter**. This hint is supported only by clusters of version 8.3.0 or later.

Precautions

- Currently, only the new version hstore tables are supported (the table-level parameter **enable_hstore_opt** is set to **on**).

Syntax

```
/* + ( no ) dict(table (column)) */
```

Parameter description

- dict(table (column))**
Column with the dictionary encoding table enabled.
- no dict(table (column))**
Column with the dictionary encoding table disabled.

Example

```
SELECT /*+ dict (bitmaptbl_high (server_ip)) */ distinct(server_ip) FROM bitmaptbl_high WHERE  
scope_name='saetataetaeta' ORDER BY server_ip;
```

The generated plan is as follows. `server_ip` uses the dictionary encoding:

```
postgres=# explain (verbose on, costs off, nodes off)  
select distinct(server_ip) from bitmaptbl_high where scope_name='saetataetaeta' order by server_ip;  
               QUERY PLAN  
-----  
id | operation  
---+-----  
1 | -> Row Adapter  
2 | -> Vector Sort  
3 | -> Vector Sonic Hash Aggregate  
4 | -> Vector Streaming (type: GATHER)  
5 | -> Vector Sonic Hash Aggregate  
6 | -> CStore Scan on public.bitmaptbl_high  
  
Predicate Information (identified by plan id)  
-----  
6 --CStore Scan on public.bitmaptbl_high  
  CU Predicate Filter: (bitmaptbl_high.scope_name = 'saetataetaeta'::text)  
  Pushdown Predicate Filter: (bitmaptbl_high.scope_name = 'saetataetaeta'::text)  
  
Targetlist Information (identified by plan id)  
-----  
1 --Row Adapter  
  Output: server_ip  
2 --Vector Sort  
  Output: server_ip  
  Sort Key: bitmaptbl_high.server_ip  
3 --Vector Sonic Hash Aggregate  
  Output: server_ip  
  Group By Key: bitmaptbl_high.server_ip  
4 --Vector Streaming (type: GATHER)  
  Output: server_ip  
5 --Vector Sonic Hash Aggregate  
  Output: server_ip  
  Group By Key: bitmaptbl_high.server_ip  
  Dict Optimized: bitmaptbl_high.server_ip  
  Dict Decoded: bitmaptbl_high.server_ip  
6 --CStore Scan on public.bitmaptbl_high  
  Output: server_ip  
  Dict Encoded: server_ip  
  Distribute Key: scope_name
```

You can use `no dict to` to disable `server_ip` from using dictionary encoding.

```
SELECT /*+ no dict (bitmaptbl_high (server_ip)) */ distinct(server_ip) FROM bitmaptbl_high WHERE  
scope_name='saetataetaeta' ORDER BY server_ip;
```

16.4.9.12 Configuration Parameter Hints

Function

A hint, or a GUC hint, specifies a configuration parameter value when a plan is generated.

Precautions

- If a parameter set by hint takes effect at the statement level, the hint must be written to the top-level query instead of the subquery. For **UNION**, **INTERSECT**, **EXCEPT**, and **MINUS** statements, you can write the GUC hint at the statement level to any **SELECT** clause that participates in the set operation. The configuration parameters set by the GUC hint take effect on each **SELECT** clause that participates in the set operation.
- When a subquery is pulled up, all GUC hints on the subquery are discarded.
- If a parameter is set by both the statement-level GUC hint and the subquery-level GUC hint, the subquery-level GUC hint takes effect in the corresponding subquery, and the statement-level GUC hint takes effect in other subqueries of the statement.

Syntax

```
set [global]([@block_name] guc_name guc_value)
```

Parameters

- **global** indicates that the parameter set by hint takes effect at the statement level. If **global** is not specified, the parameter takes effect only in the subquery where the hint is located.
- *block_name* indicates the block name of the statement block. For details, see [block_name](#).
- **guc_name** indicates the name of the configuration parameter specified by hint.
- **guc_value** indicates the value of a configuration parameter specified by hint.

Currently, GUC hints support only some configuration parameters. Some parameters cannot be configured at the subquery level and can only be configured at the statement level. The following table lists the supported parameters.

Table 16-19 Configuration parameters supported by GUC hints

| Parameter | Configured at the Subquery Level (Yes/No) |
|------------------------------|---|
| agg_max_mem | Yes |
| agg_redistribute_enhancement | Yes |

| Parameter | Configured at the Subquery Level (Yes/No) |
|----------------------------------|---|
| best_agg_plan | Yes |
| cost_model_version | No |
| cost_param | No |
| enable_array_optimization | No |
| enable_bitmapscan | Yes |
| enable_broadcast | Yes |
| enable_csqual_pushdown | No |
| enable_redistribute | Yes |
| enable_extrapolation_stats | Yes |
| enable_fast_query_shipping | No |
| enable_force_vector_engine | No |
| enable_hashagg | Yes |
| enable_hashfilter | No |
| enable_hashjoin | Yes |
| enable_index_nestloop | Yes |
| enable_indexonlyscan | Yes |
| enable_indexscan | Yes |
| enable_join_pseudoconst | Yes |
| enable_mergejoin | Yes |
| enable_mixedagg | No |
| enable_nestloop | Yes |
| enable_nodegroup_debug | No |
| enable_partition_dynamic_pruning | Yes |
| enable_seqscan | Yes |
| enable_sonic_hashagg | No |
| enable_sonic_hashjoin | Yes |
| enable_sort | Yes |
| enable_stream_ctescan | No |
| enable_tidscan | Yes |

| Parameter | Configured at the Subquery Level (Yes/No) |
|-----------------------------------|---|
| enable_value_redistribute | Yes |
| enable_vector_engine | No |
| expected_computing_nodegroup | No |
| force_bitmapand | Yes |
| from_collapse_limit | Yes |
| join_collapse_limit | Yes |
| join_num_distinct | Yes |
| outer_join_max_rows_multipler | Yes |
| prefer_hashjoin_path | No |
| qrw_inlist2join_optmode | Yes |
| qual_num_distinct | Yes |
| query_dop | No |
| query_max_mem | No |
| query_mem | No |
| rewrite_rule | No |
| setop_optmode | Yes |
| skew_option | Yes |
| stream_ctescan_max_estimate_mem | No |
| stream_ctescan_pred_threshold | No |
| stream_ctescan_refcount_threshold | No |
| windowagg_pushdown_enhancement | No |
| index_selectivity_cost | Yes |
| index_cost_limit | Yes |

Examples

Hint the query plan in [Examples](#) as follows:

```
explain
select /*+ set global(query_dop 0) */ i_product_name product_name
...
```

This hint indicates that the **query_dop** parameter is set to **0** when the plan for a statement is generated, which means the SMP adaptation function is enabled. The generated plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|-----------|--|---------|----------|---------|----------|
| 1 | -> Row Adapter | 1 | | 230 | 19595.89 |
| 2 | -> Vector Sonic Hash Aggregate | 1 | | 230 | 19595.89 |
| 3 | -> Vector Streaming (type: GATHER) | 3 | | 230 | 19595.89 |
| 4 | -> Vector Sonic Hash Aggregate | 3 | 16MB | 230 | 19595.66 |
| 5 | -> Vector Nest Loop (6,28) | 3 | 1MB | 126 | 19595.62 |
| 6 | -> Vector Nest Loop (7,27) | 3 | 1MB | 130 | 19291.57 |
| 7 | -> Vector Streaming(type: LOCAL GATHER dop: 1/2) | 3 | 4MB | 118 | 19279.41 |
| 8 | -> Vector Nest Loop (9,24) | 3 | 1MB | 118 | 19279.38 |
| 9 | -> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2) | 3 | 4MB | 82 | 18117.66 |
| 10 | -> Vector Nest Loop (11,21) | 3 | 1MB | 82 | 18117.61 |
| 11 | -> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 2/2) | 3 | 4MB | 82 | 16195.20 |
| 12 | -> Vector Sonic Hash Join (13,15) | 3 | 16MB | 82 | 16195.15 |
| 13 | -> Vector Partition Iterator | 287514 | 1MB | 12 | 1110.42 |
| 14 | -> Partitioned CStore Scan on store_returns | 287514 | 1MB | 12 | 1110.42 |
| 15 | -> Vector Streaming(type: LOCAL BROADCAST dop: 2/2) | 2764 | 4MB | 94 | 14718.42 |
| 16 | -> Vector Sonic Hash Join (17,19) | 1382 | 16MB | 94 | 14699.69 |
| 17 | -> Vector Partition Iterator | 2880404 | 1MB | 39 | 11541.07 |
| 18 | -> Partitioned CStore Scan on store_sales | 2880404 | 1MB | 39 | 11541.07 |
| 19 | -> Vector Streaming(type: LOCAL BROADCAST dop: 2/2) | 16 | 4MB | 55 | 1947.12 |
| 20 | -> CStore Scan on item | 8 | 1MB | 55 | 1947.00 |
| 21 | -> Vector Materialize | 100000 | 16MB | 8 | 1797.41 |
| 22 | -> Vector Streaming(type: LOCAL REDISTRIBUTE dop: 2/2) | 100000 | 4MB | 8 | 1714.07 |
| 23 | -> CStore Scan on customer | 100000 | 1MB | 8 | 703.67 |
| 24 | -> Vector Materialize | 50000 | 16MB | 44 | 1099.22 |
| 25 | -> Vector Streaming(type: LOCAL REDISTRIBUTE dop: 2/2) | 50000 | 4MB | 44 | 1057.55 |
| 26 | -> CStore Scan on customer_address ad2 | 50000 | 1MB | 44 | 552.33 |
| 27 | -> CStore Scan on store | 36 | 1MB | 20 | 12.01 |
| 28 | -> CStore Scan on promotion | 900 | 1MB | 4 | 300.30 |
| (28 rows) | | | | | |

16.4.9.13 Hint Errors, Conflicts, and Other Warnings

Plan hints change an execution plan. You can run **EXPLAIN** to view the changes.

Hints containing errors are invalid and do not affect statement execution. The errors will be displayed in different ways based on statement types. Hint errors in an **EXPLAIN** statement are displayed as a warning on the interface. Hint errors in other statements will be recorded in debug1-level logs containing the **PLANHINT** keyword.

Hint Error Types

- Syntax errors.

An error will be reported if the syntax tree fails to be reduced. The No. of the row generating an error is displayed in the error details.

For example, the hint keyword is incorrect, no table or only one table is specified in the **leading** or **join** hint, or no tables are specified in other hints. The parsing of a hint is terminated immediately after a syntax error is detected. Only the hints that have been parsed successfully are valid.

For example:

```
leading((t1 t2)) nestloop(t1) rows(t1 t2 #10)
```

The syntax of **nestloop(t1)** is wrong and its parsing is terminated. Only **leading(t1 t2)** that has been successfully parsed before **nestloop(t1)** is valid.

- Semantic errors.
 - An error will be reported if the specified tables do not exist, multiple tables are found based on the hint setting, or a table is used more than once in the **leading** or **join** hint.
 - An error will be reported if the index specified in a scan hint does not exist.
 - If multiple tables with the same name exist after a subquery is pulled up and some of them need to be hinted, add aliases for them to avoid name duplication.
- Duplicated or conflicted hints.

If hint duplication or conflicts occur, only the first hint takes effect. A message will be displayed to describe the situation.

- Hint duplication indicates that a hint is used more than once in the same query, for example, **nestloop(t1 t2) nestloop(t1 t2)**.
- A hint conflict indicates that the functions of two hints with the same table list conflict with each other.

For example, if **nestloop (t1 t2) hashjoin (t1 t2)** is used, **hashjoin (t1 t2)** becomes invalid. **nestloop(t1 t2)** does not conflict with **no mergejoin(t1 t2)**.

NOTICE

The table list in the **leading** hint is disassembled. For example, **leading (t1 t2 t3)** will be disassembled as **leading(t1 t2) leading((t1 t2) t3)**, which will conflict with **leading(t2 t1)** (if any). In this case, the latter **leading(t2 t1)** becomes invalid. If two hints use duplicated table lists and only one of them has the specified outer/inner table, the one without a specified outer/inner table becomes invalid.

- A hint becomes invalid after a sublink is pulled up.
In this case, a message will be displayed. Generally, such invalidation occurs if a sublink contains multiple tables to be joined, because the table list in the sublink becomes invalid after the sublink is pulled up.
- Unsupported column types.
 - Skew hints are specified to optimize redistribution. They will be invalid if their corresponding columns do not support redistribution.
- Specified hints are not used.
 - If **hashjoin** or **mergejoin** is specified for non-equivalent joins, it will not be used.
 - If **indexscan** or **indexonlyscan** is specified for a table that does not have an index, it will not be used.
 - If **indexscan hint** or **indexonlyscan** is specified for a full-table scan or for a scan whose filtering conditions are not set on index columns, it will not be used.
 - The specified **indexonlyscan** hint is used only when the output column contains only indexes.
 - In equivalent joins, only the joins containing equivalence conditions are valid. Therefore, the **leading**, **join**, and **rows** hints specified for the joins without an equivalence condition will not be used. For example, **t1**, **t2**, and **t3** are to be joined, and the join between **t1** and **t3** does not contain an equivalence condition. In this case, **leading(t1 t3)** will not be used.
 - To generate a streaming plan, if the distribution key of a table is the same as its join key, **redistribute** specified for this table will not be used. If the distribution key and join key are different for this table but the same for the other table in the join, **redistribute** specified for this table will be used but **broadcast** will not.
 - If a hint for an **Agg** distribution column is not used, the possible causes are as follows:

- The specified distribution key contains data types that do not support redistribution.
- Redistribution is not required in the execution plan.
- Wrong distribution key sequence numbers are executed.
- For AP functions that use the GROUPING SETS and CUBE clauses, hints are not supported for distribution keys in window aggregate functions .

** NOTE**

Specifies the hint for the distribution column during the Agg process.. This parameter is supported only by clusters of version 8.1.3.100 or later.

- If no sublink is pulled up, the specified **blockname** hint will not be used.
- For unused skew hints, the possible causes are:
 - The plan does not require redistribution.
 - The columns specified by hints contain distribution keys.
 - Skew information specified in hints is incorrect or incomplete, for example, no value is specified for join optimization.
 - Skew optimization is disabled by GUC parameters.
- For unused guc hints, the possible causes are:
 - The configuration parameter does not exist.
 - The configuration parameter is not supported by GUC hints.
 - The configuration parameter value is invalid.
 - The statement-level GUC hint is not written in the top-level query.
 - The configuration parameter set by the GUC hint at the subquery level cannot be set at the subquery level.
 - The subquery where the GUC hint is located is pulled up.

16.4.9.14 Plan Hint Cases

This section takes the statements in TPC-DS (Q24) as an example to describe how to optimize an execution plan by using hints in 1000X+24DN environments. For example:

```
select avg(netpaid) from
(select c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size
```

```
,sum(ss_sales_price) netpaid
from store_sales
,store_returns
,store
,item
,customer
,customer_address
where ss_ticket_number = sr_ticket_number
and ss_item_sk = sr_item_sk
and ss_customer_sk = c_customer_sk
and ss_item_sk = i_item_sk
and ss_store_sk = s_store_sk
and c_birth_country = upper(ca_country)
and s_zip = ca_zip
and s_market_id=7
group by c_last_name
,c_first_name
,s_store_name
,ca_state
,s_state
,i_color
,i_current_price
,i_manager_id
,i_units
,i_size);
```

1. The original plan of this statement is as follows and the statement execution takes 110s:

Figure 16-10 Statement initial plan

| id | operation | A-time | A-rows | E-rows |
|----|--|-----------------------|------------|------------|
| 1 | -> Row Adapter | 110324.107 | 1 | 1 |
| 2 | -> Vector Aggregate | 110324.093 | 1 | 1 |
| 3 | -> Vector Streaming (type: GATHER) | 110323.958 | 24 | 24 |
| 4 | -> Vector Aggregate | 110178.302,110309.653 | 24 | 24 |
| 5 | -> Vector Hash Aggregate | 110178.388,110308.515 | 647824 | 16656 |
| 6 | -> Vector Streaming (type: REDISTRIBUTE) | 177616.177,96478.771 | 666834733 | 16664 |
| 7 | -> Vector Hash Join (8,22) | 81727.257,84728.519 | 666834733 | 16664 |
| 8 | -> Vector Streaming (type: REDISTRIBUTE) | 78770.520,82021.087 | 666834733 | 16664 |
| 9 | -> Vector Hash Join (10,21) | 88066.755,90701.860 | 666834733 | 16664 |
| 10 | -> Vector Streaming (type: BROADCAST) | 7940.962,21430.725 | 591882336 | 51360 |
| 11 | -> Vector Hash Join (12,20) | 2419.995,5319.606 | 24661764 | 2140 |
| 12 | -> Vector Streaming (type: REDISTRIBUTE) | 1750.446,4659.581 | 25259268 | 2241 |
| 13 | -> Vector Hash Join (14,18) | 15240.666,17159.616 | 25259268 | 2241 |
| 14 | -> Vector Hash Join (15,17) | 12112.913,13563.366 | 252564412 | 472070592 |
| 15 | -> Vector Partition Iterator | 11148.731,12473.230 | 2879987999 | 2879987999 |
| 16 | -> Partitioned CStore Scan on public.store_sales | 11097.921,12412.596 | 2879987999 | 2879987999 |
| 17 | -> CStore Scan on public.store | 0.447,0.689 | 2064 | 2064 |
| 18 | -> Vector Partition Iterator | 296.805,319.014 | 287999764 | 287999764 |
| 19 | -> Partitioned CStore Scan on public.store_returns | 292.938,314.787 | 287999764 | 287999764 |
| 20 | -> CStore Scan on public.customer | 114.358,144.462 | 12000000 | 12000000 |
| 21 | -> CStore Scan on public.customer_address | 38.426,56.753 | 6000000 | 6000000 |
| 22 | -> CStore Scan on public.item | 3.160,5.026 | 300000 | 300000 |

(22 rows)

In this plan, the performance of the layer-10 **broadcast** is poor because the number of rows estimated by the layer-11 operator is 2,140, which is much lower than the actual number of rows. The inaccurate estimation is mainly caused by the underestimated number of rows in layer-13 hash join. In this layer, **store_sales** and **store_returns** are joined (based on the **ss_ticket_number** and **ss_item_sk** columns in **store_sales** and the **sr_ticket_number** and **sr_item_sk** columns in **store_returns**) but the multi-column correlation is not considered.

2. After the **rows** hint is used for optimization, the plan is as follows and the statement execution takes 318s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns * 11270)*/ c_last_name ...
```

Figure 16-11 Using rows hints for optimization

| id | operation | A-time | A-rows | E-rows |
|----|--|--------------------------|------------|------------|
| 1 | -> Row Adapter | 318585.246 | 1 | 1 |
| 2 | -> Vector Aggregate | 318585.232 | 1 | 1 |
| 3 | -> Vector Streaming (type: GATHER) | 318585.082 | 24 | 24 |
| 4 | -> Vector Aggregate | [318323.324, 318499.290] | 24 | 24 |
| 5 | -> Vector Hash Aggregate | [318320.813, 318497.054] | 647824 | 187770504 |
| 6 | -> Vector Streaming(type: REDISTRIBUTE) | [288074.860, 305601.698] | 666834733 | 187770507 |
| 7 | -> Vector Hash Join (8,22) | [253642.468, 315808.664] | 666834733 | 187770507 |
| 8 | -> Vector Hash Join (9,18) | [250904.317, 315684.018] | 666834733 | 187770507 |
| 9 | -> Vector Streaming(type: REDISTRIBUTE) | [4552.500, 310602.307] | 275042158 | 147106999 |
| 10 | -> Vector Hash Join (11,17) | [7659.951, 14053.823] | 275042158 | 147106999 |
| 11 | -> Vector Streaming(type: REDISTRIBUTE) | [3953.255, 10264.943] | 287999764 | 154060900 |
| 12 | -> Vector Hash Join (13,15) | [28196.188, 32838.794] | 287999764 | 154060900 |
| 13 | -> Vector Partition Iterator | [11477.673, 12324.583] | 2879987999 | 2879987999 |
| 14 | -> Partitioned CStore Scan on public.store_sales | [11411.382, 12250.209] | 2879987999 | 2879987999 |
| 15 | -> Vector Partition Iterator | [304.188, 403.205] | 287999764 | 287999764 |
| 16 | -> Partitioned CStore Scan on public.store_returns | [299.838, 398.255] | 287999764 | 287999764 |
| 17 | -> CStore Scan on public.customer | [122.246, 170.128] | 12000000 | 12000000 |
| 18 | -> Vector Streaming(type: REDISTRIBUTE) | [57.558, 117.461] | 492915 | 146467 |
| 19 | -> Vector Hash Join (20,21) | [45.554, 96.238] | 492915 | 146467 |
| 20 | -> CStore Scan on public.customer_address | [39.738, 89.412] | 6000000 | 6000000 |
| 21 | -> CStore Scan on public.store | [0.361, 1.095] | 2064 | 2064 |
| 22 | -> Vector Streaming(type: BROADCAST) | [48.986, 91.170] | 7200000 | 7200000 |
| 23 | -> CStore Scan on public.item | [4.506, 6.602] | 300000 | 300000 |

(23 rows)

The execution takes a longer time because layer-9 **redistribute** is slow. Considering that data skew does not occur at layer-9 **redistribute**, the slow redistribution is caused by the slow layer-8 **hashjoin** due to data skew at layer-18 **redistribute**.

3. Data skew occurs at layer-18 **redistribute** because **customer_address** has a few different values in its two join keys. Therefore, plan **customer_address** as the last one to be joined. After the hint is used for optimization, the plan is as follows and the statement execution takes 116s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((store_sales store_returns store item customer) customer_address)*/
c_last_name ...
```

Figure 16-12 Hint optimization

| id | operation | A-time | A-rows | E-rows |
|----|--|--------------------------|------------|------------|
| 1 | -> Row Adapter | 116326.597 | 1 | 1 |
| 2 | -> Vector Aggregate | 116326.590 | 1 | 1 |
| 3 | -> Vector Streaming (type: GATHER) | 116326.473 | 24 | 24 |
| 4 | -> Vector Aggregate | [116157.161, 116236.494] | 24 | 24 |
| 5 | -> Vector Hash Aggregate | [116155.328, 116233.946] | 647824 | 187770504 |
| 6 | -> Vector Streaming(type: REDISTRIBUTE) | [84103.951, 102052.326] | 666834733 | 187770507 |
| 7 | -> Vector Hash Join (8,10) | [23229.469, 47484.697] | 666834733 | 187770507 |
| 8 | -> Vector Streaming(type: REDISTRIBUTE) | [38.367, 74.930] | 6000000 | 6000000 |
| 9 | -> CStore Scan on public.customer_address | [69.877, 121.460] | 6000000 | 6000000 |
| 10 | -> Vector Streaming(type: REDISTRIBUTE) | [17404.744, 17587.550] | 24661764 | 24112909 |
| 11 | -> Vector Hash Join (12,22) | [16123.627, 16397.246] | 24661764 | 24112909 |
| 12 | -> Vector Streaming(type: REDISTRIBUTE) | [15320.663, 15741.646] | 25258268 | 25252751 |
| 13 | -> Vector Hash Join (14,21) | [14962.342, 16375.458] | 25258268 | 25252751 |
| 14 | -> Vector Hash Join (15,19) | [14449.031, 15825.949] | 25258268 | 25252751 |
| 15 | -> Vector Hash Join (16,18) | [11439.959, 12510.065] | 252564412 | 472070592 |
| 16 | -> Vector Partition Iterator | [10531.986, 11536.213] | 2879987999 | 2879987999 |
| 17 | -> Partitioned CStore Scan on public.store_sales | [10483.634, 11474.944] | 2879987999 | 2879987999 |
| 18 | -> CStore Scan on public.store | [0.347, 0.463] | 2064 | 2064 |
| 19 | -> Vector Partition Iterator | [293.977, 365.021] | 287999764 | 287999764 |
| 20 | -> Partitioned CStore Scan on public.store_returns | [289.936, 360.808] | 287999764 | 287999764 |
| 21 | -> CStore Scan on public.item | [3.109, 5.245] | 300000 | 300000 |
| 22 | -> CStore Scan on public.customer | [113.871, 141.791] | 12000000 | 12000000 |

(22 rows)

Most of the time is spent on layer-6 **redistribute**. The plan needs to be further optimized.

4. The last layer redistribute contains skew. Therefore, it takes a long time. To avoid the data skew, plan the **item** table as the last one to be joined because the number of rows is not reduced after **item** is joined. After the hint is used for optimization, the plan is as follows and the statement execution takes 120s:

```
select avg(netpaid) from
(select /*+rows(store_sales store_returns *11270)
leading((customer_address (store_sales store_returns store customer) item))
c_last_name ...
```

Figure 16-13 Modifying hints and executing statements

| id | operation | A-time | A-rows | E-rows |
|----|--|-------------------------|------------|------------|
| 1 | -> Row Adapter | 120377.258 | 1 | 1 |
| 2 | -> Vector Aggregate | 120377.245 | 1 | 1 |
| 3 | -> Vector Streaming (type: GATHER) | 120377.091 | 24 | 24 |
| 4 | -> Vector Aggregate | [120184.884,120301.704] | 24 | 24 |
| 5 | -> Vector Hash Aggregate | [120183.119,120297.845] | 647824 | 187770504 |
| 6 | -> Vector Streaming (type: REDISTRIBUTE) | [87775.682,106070.815] | 666834733 | 187770507 |
| 7 | -> Vector Hash Join (8,22) | [23223.764,49878.523] | 666834733 | 187770507 |
| 8 | -> Vector Hash Join (9,11) | [21129.236,45208.255] | 666834733 | 187770507 |
| 9 | -> Vector Streaming (type: REDISTRIBUTE) | [37.859,75.412] | 6000000 | 6000000 |
| 10 | -> CStore Scan on public.customer_address | [74.798,114.449] | 6000000 | 6000000 |
| 11 | -> Vector Streaming (type: REDISTRIBUTE) | [15714.458,15824.928] | 24661764 | 24112909 |
| 12 | -> Vector Hash Join (13,21) | [14637.516,14955.464] | 24661764 | 24112909 |
| 13 | -> Vector Streaming (type: REDISTRIBUTE) | [13898.593,14333.200] | 25258268 | 25252751 |
| 14 | -> Vector Hash Join (15,19) | [14166.917,15378.244] | 25258268 | 25252751 |
| 15 | -> Vector Hash Join (16,18) | [11272.239,12052.532] | 252564412 | 472070592 |
| 16 | -> Vector Partition Iterator | [10409.566,11127.981] | 2879987999 | 2879987999 |
| 17 | -> Partitioned CStore Scan on public.store_sales | [10365.838,11077.601] | 2879987999 | 2879987999 |
| 18 | -> CStore Scan on public.store | [0.431,0.609] | 2064 | 2064 |
| 19 | -> Vector Partition Iterator | [343.780,408.254] | 287999764 | 287999764 |
| 20 | -> Partitioned CStore Scan on public.store_returns | [339.844,403.923] | 287999764 | 287999764 |
| 21 | -> CStore Scan on public.customer | [117.234,163.598] | 12000000 | 12000000 |
| 22 | -> Vector Streaming (type: BROADCAST) | [44.571,130.129] | 7200000 | 7200000 |
| 23 | -> CStore Scan on public.item | [4.169,6.347] | 300000 | 300000 |

(23 rows)

Data skew occurs after the join of **item** and **customer_address** because **item** is broadcasted at layer-22. As a result, layer-6 **redistribute** is still slow.

5. Add a hint to disable **broadcast** for **item** or add a **redistribute** hint for the join result of **item** and **customer_address**. After the hint is used for optimization, the plan is as follows and the statement execution takes 105s:

```
select avg(netpaid) from  
(select /*+rows(store_sales store_returns *11270)  
leading((customer_address (store_sales store_returns store customer) item))  
no broadcast(item)*/  
c_last_name ...
```

Figure 16-14 Execution plan

| id | operation | A-time | A-rows | E-rows |
|----|--|-------------------------|------------|------------|
| 1 | -> Row Adapter | 105854.957 | 1 | 1 |
| 2 | -> Vector Aggregate | 105854.948 | 1 | 1 |
| 3 | -> Vector Streaming (type: GATHER) | 105854.825 | 24 | 24 |
| 4 | -> Vector Aggregate | [105706.709,105776.135] | 24 | 24 |
| 5 | -> Vector Hash Aggregate | [105705.061,105773.013] | 647824 | 187770504 |
| 6 | -> Vector Streaming (type: REDISTRIBUTE) | [70701.966,89973.672] | 666834733 | 187770507 |
| 7 | -> Vector Hash Join (8,23) | [71759.500,79018.433] | 666834733 | 187770507 |
| 8 | -> Vector Streaming (type: REDISTRIBUTE) | [69794.307,77269.178] | 666834733 | 187770507 |
| 9 | -> Vector Hash Join (10,12) | [21443.307,46714.378] | 666834733 | 187770507 |
| 10 | -> Vector Streaming (type: REDISTRIBUTE) | [41.295,83.419] | 6000000 | 6000000 |
| 11 | -> CStore Scan on public.customer_address | [70.405,166.072] | 6000000 | 6000000 |
| 12 | -> Vector Streaming (type: REDISTRIBUTE) | [15689.053,15788.475] | 24661764 | 24112909 |
| 13 | -> Vector Hash Join (14,22) | [14517.847,14712.929] | 24661764 | 24112909 |
| 14 | -> Vector Streaming (type: REDISTRIBUTE) | [13806.733,14089.770] | 25258268 | 25252751 |
| 15 | -> Vector Hash Join (16,20) | [13709.384,15095.449] | 25258268 | 25252751 |
| 16 | -> Vector Hash Join (17,19) | [10944.796,11827.285] | 252564412 | 472070592 |
| 17 | -> Vector Partition Iterator | [10070.316,10884.728] | 2879987999 | 2879987999 |
| 18 | -> Partitioned CStore Scan on public.store_sales | [10018.966,10828.990] | 2879987999 | 2879987999 |
| 19 | -> CStore Scan on public.store | [0.447,0.568] | 2064 | 2064 |
| 20 | -> Vector Partition Iterator | [293.042,329.056] | 287999764 | 287999764 |
| 21 | -> Partitioned CStore Scan on public.store_returns | [288.631,324.782] | 287999764 | 287999764 |
| 22 | -> CStore Scan on public.customer | [113.735,138.235] | 12000000 | 12000000 |
| 23 | -> CStore Scan on public.item | [3.127,5.357] | 300000 | 300000 |

(23 rows)

6. The last layer uses single-layer **Agg** and the number of rows is greatly reduced. Set **best_agg_plan** to **3** and change the single-layer **Agg** to a double-layer **Agg**. The plan is as follows and the statement execution takes 94s. The optimization ends.

Figure 16-15 Final optimization plan

| id | operation | A-time | A-rows | E-rows |
|----|--|-----------------------|------------|------------|
| 1 | -> Row Adapter | 94004.670 | 1 | 1 |
| 2 | -> Vector Aggregate | 94004.655 | 1 | 1 |
| 3 | -> Vector Streaming (type: GATHER) | 94004.504 | 24 | 24 |
| 4 | -> Vector Aggregate | [93933.852,93928.052] | 24 | 24 |
| 5 | -> Vector Hash Aggregate | [93832.460,93926.412] | 647824 | 187770507 |
| 6 | -> Vector Streaming(type: REDISTRIBUTE) | [93640.866,93787.939] | 647824 | 183912384 |
| 7 | -> Vector Hash Aggregate | [93687.544,93791.242] | 647824 | 183912384 |
| 8 | -> Vector Hash Join (9,24) | [70025.469,72773.161] | 666834733 | 187770507 |
| 9 | -> Vector Streaming(type: REDISTRIBUTE) | [68242.223,71275.972] | 666834733 | 187770507 |
| 10 | -> Vector Hash Join (11,13) | [21421.136,44830.306] | 666834733 | 187770507 |
| 11 | -> Vector Streaming(type: REDISTRIBUTE) | [35.444,71.328] | 6000000 | 6000000 |
| 12 | -> CStore Scan on public.customer_address | [67.246,119.224] | 6000000 | 6000000 |
| 13 | -> Vector Streaming(type: REDISTRIBUTE) | [16089.853,16212.570] | 24661764 | 24112909 |
| 14 | -> Vector Hash Join (15,23) | [14822.972,15188.942] | 24661764 | 24112909 |
| 15 | -> Vector Streaming(type: REDISTRIBUTE) | [14061.867,14604.162] | 25258268 | 25252751 |
| 16 | -> Vector Hash Join (17,21) | [13949.756,15492.311] | 25258268 | 25252751 |
| 17 | -> Vector Hash Join (18,20) | [10935.742,12160.719] | 252564412 | 472070592 |
| 18 | -> Vector Partition Iterator | [10052.958,11194.962] | 2879987999 | 2879987999 |
| 19 | -> Partitioned CStore Scan on public.store_sales | [10008.415,11143.984] | 2879987999 | 2879987999 |
| 20 | -> CStore Scan on public.store | [0.452,0.839] | 2064 | 2064 |
| 21 | -> Vector Partition Iterator | [298.235,332.736] | 287999764 | 287999764 |
| 22 | -> Partitioned CStore Scan on public.store_returns | [294.067,327.629] | 287999764 | 287999764 |
| 23 | -> CStore Scan on public.customer | [114.377,145.156] | 12000000 | 12000000 |
| 24 | -> CStore Scan on public.item | [3.150,3.530] | 300000 | 300000 |

(24 rows)

If the query performance deteriorates due to statistics changes, you can use hints to optimize the query plan. Take TPC-H-Q17 as an example. The query performance deteriorates after the value of **default_statistics_target** is changed from the default one to -2 for statistics collection.

1. If **default_statistics_target** is set to the default value **100**, the plan is as follows:

Figure 16-16 Default statistics

| id | operation | A-time |
|----|--|-------------------------|
| 1 | -> Row Adapter | 265006.779 |
| 2 | -> Vector Aggregate | 265006.764 |
| 3 | -> Vector Streaming (type: GATHER) | 265006.071 |
| 4 | -> Vector Aggregate | [263699.512,264503.084] |
| 5 | -> Vector Hash Join (6,17) | [263676.665,264477.932] |
| 6 | -> Vector Streaming(type: LOCAL GATHER dop: 1/4) | [1.998,7.594] |
| 7 | -> Vector Hash Aggregate | [201775.393,202432.672] |
| 8 | -> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 4/4) | [201567.130,202231.524] |
| 9 | -> Vector Hash Join (10,12) | [170675.231,199908.410] |
| 10 | -> Vector Partition Iterator | [34847.797,51968.266] |
| 11 | -> Partitioned CStore Scan on tpch10wx_col.lineitem | [33805.019,51137.657] |
| 12 | -> Vector Hash Aggregate | [23283.387,25359.493] |
| 13 | -> Vector Streaming(type: SPLIT BROADCAST dop: 4/4) | [12850.624,14608.515] |
| 14 | -> Vector Hash Aggregate | [2690.439,3616.623] |
| 15 | -> Vector Partition Iterator | [2659.700,3579.390] |
| 16 | -> Partitioned CStore Scan on tpch10wx_col.part | [2642.213,3559.093] |
| 17 | -> Vector Streaming(type: REDISTRIBUTE dop: 1/4) | [262300.732,262961.078] |
| 18 | -> Vector Hash Join (19,21) | [225749.727,260990.322] |
| 19 | -> Vector Partition Iterator | [40046.078,56220.694] |
| 20 | -> Partitioned CStore Scan on tpch10wx_col.lineitem | [39204.414,55328.448] |
| 21 | -> Vector Streaming(type: SPLIT BROADCAST dop: 4/4) | [55748.177,61987.136] |
| 22 | -> Vector Partition Iterator | [3042.864,3873.942] |
| 23 | -> Partitioned CStore Scan on tpch10wx_col.part | [3027.023,3848.159] |

(23 rows)

2. If **default_statistics_target** is set to -2, the plan is as follows.

Figure 16-17 Changes in statistics

| id | operation | A-time |
|----|--|---------------------------|
| 1 | -> Row Adapter | 1440492.994 |
| 2 | -> Vector Aggregate | 1440492.982 |
| 3 | -> Vector Streaming (type: GATHER) | 1440491.021 |
| 4 | -> Vector Streaming(type: LOCAL GATHER dop: 1/6) | [1439737.284,1440008.568] |
| 5 | -> Vector Aggregate | [1439008.369,1439854.148] |
| 6 | -> Vector Hash Join (7,18) | [1439006.016,1439851.619] |
| 7 | -> Vector Streaming(type: LOCAL BROADCAST dop: 6/6) | [2.932,139.405] |
| 8 | -> Vector Hash Aggregate | [190452.312,195910.748] |
| 9 | -> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6) | [190171.929,195653.119] |
| 10 | -> Vector Hash Join (11,13) | [161076.195,178831.123] |
| 11 | -> Vector Partition Iterator | [27306.318,45564.565] |
| 12 | -> Partitioned CStore Scan on tpch10wx_col.lineitem | [26752.444,44912.020] |
| 13 | -> Vector Hash Aggregate | [35601.624,39812.058] |
| 14 | -> Vector Streaming(type: SPLIT BROADCAST dop: 6/6) | [23096.460,27057.137] |
| 15 | -> Vector Hash Aggregate | [2372.587,3052.445] |
| 16 | -> Vector Partition Iterator | [2345.381,3012.732] |
| 17 | -> Partitioned CStore Scan on tpch10wx_col.part | [2329.874,2989.393] |
| 18 | -> Vector Hash Join (19,22) | [1437388.414,1438470.781] |
| 19 | -> Vector Streaming(type: SPLIT REDISTRIBUTE dop: 6/6) | [1392693.529,1408571.859] |
| 20 | -> Vector Partition Iterator | [23065.204,4264.514] |
| 21 | -> Partitioned CStore Scan on tpch10wx_col.lineitem | [28212.219,40133.491] |
| 22 | -> Vector Streaming(type: LOCAL REDISTRIBUTE dop: 6/6) | [2570.841,3438.567] |
| 23 | -> Vector Partition Iterator | [2447.569,3276.369] |
| 24 | -> Partitioned CStore Scan on tpch10wx_col.part | [2432.124,3263.641] |

(24 rows)

3. After the analysis, the cause is that the stream type is changed from **BroadCast** to **Redistribute** during the join of the **lineitem** and **part** tables. You can use a hint to change the stream type back to **BroadCast**. The figure below shows an example.

Figure 16-18 Statements

```
select /*+ no redistribute(part lineitem) */
      sum(l_extendedprice) / 7.0 as avg_yearly
from
      lineitem,
      part
where
      p_partkey = l_partkey
      and p_brand = 'Brand#23'
      and p_container = 'MED BOX'
      and l_quantity < (
          select
              0.2 * avg(l_quantity)
          from
              lineitem
          where
              l_partkey = p_partkey
      );
```

16.4.10 Routinely Maintaining Tables

To ensure proper database running, after INSERT and DELETE operations, you need to routinely do **VACUUM FULL** and **ANALYZE** as appropriate for customer scenarios and update statistics to obtain better performance.

Related Concepts

You need to routinely run **VACUUM**, **VACUUM FULL**, and **ANALYZE** to maintain tables, because:

- **VACUUM FULL** reclaims disk space occupied by updated or deleted data and combines small-size data files.
- **VACUUM** maintains a visualized mapping to track pages that contain arrays visible to other active transactions. A common index scan uses the mapping to obtain the corresponding array and check whether pages are visible to the current transaction. If the array cannot be obtained, the visibility is checked by fetching stack arrays. Therefore, updating the visible mapping of a table can accelerate unique index scans.
- **VACUUM** can avoid old data loss caused by duplicate transaction IDs when the number of executed transactions exceeds the database threshold.
- **ANALYZE** collects statistics on tables in databases. The statistics are stored in the PG_STATISTIC system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

Procedure

Step 1 Run the **VACUUM** or **VACUUM FULL** command to reclaim disk space.

- **VACUUM:**

Do **VACUUM** to the table:

```
VACUUM customer;  
VACUUM
```

This command can be concurrently executed with database operation commands, including **SELECT**, **INSERT**, **UPDATE**, and **DELETE**; excluding **ALTER TABLE**.

Do **VACUUM** to the partitioned table:

```
VACUUM customer_par PARTITION ( P1 );  
VACUUM
```

- **VACUUM FULL:**

```
VACUUM FULL customer;  
VACUUM
```

VACUUM FULL needs to add exclusive locks on tables it operates on and requires that all other database operations be suspended.

When reclaiming disk space, you can query for the session corresponding to the earliest transactions in the cluster, and then end the earliest long transactions as needed to make full use of the disk space.

- a. Run the following command to query for oldestxmin on the GTM:

```
select * from pgxc_gtm_snapshot_status();
```
- b. Run the following command to query for the PID of the corresponding session on the CN. *xmin* is the oldestxmin obtained in the previous step.

```
select * from pgxc_running_xacts() where xmin=1400202010;
```

Step 2 Do **ANALYZE** to update statistical information.

```
ANALYZE customer;  
ANALYZE
```

Do **ANALYZE VERBOSE** to update statistics and display table information.

```
ANALYZE VERBOSE customer;  
ANALYZE
```

You can use **VACUUM ANALYZE** at the same time to optimize the query.

```
VACUUM ANALYZE customer;  
VACUUM
```

NOTE

VACUUM and **ANALYZE** cause a substantial increase in I/O traffic, which may cause poor performance of other active sessions. Therefore, you are advised to set by specifying the **vacuum_cost_delay** parameter.

Step 3 Delete a table

```
DROP TABLE customer;  
DROP TABLE customer_par;  
DROP TABLE part;
```

If the following output is displayed, the index has been deleted.

```
DROP TABLE
```

----End

Maintenance Suggestion

- Routinely do **VACUUM FULL** to large tables. If the database performance deteriorates, do **VACUUM FULL** to the entire database. If the database performance is stable, you are advised to monthly do **VACUUM FULL**.
- Routinely do **VACUUM FULL** to system catalogs, mainly **PG_ATTRIBUTE**.
- The automatic vacuum process (**AUTOVACUUM**) in the system automatically runs the **VACUUM** and **ANALYZE** statements to reclaim the record space marked as the deleted state and to update statistics related to the table.

16.4.11 Routinely Recreating an Index

Context

When data deletion is repeatedly performed in the database, index keys will be deleted from the index page, resulting in index distention. Recreating an index routinely improves query efficiency.

The database supports B-tree, GIN, and psort indexes.

- Recreating a B-tree index helps improve query efficiency.
 - If massive data is deleted, index keys on the index page will be deleted. As a result, the number of index pages reduces and index bloat occurs. Recreating an index helps reclaim wasted space.
 - In the created index, pages adjacent in its logical structure are adjacent in its physical structure. Therefore, a created index achieves higher access speed than an index that has been updated for multiple times.
- You are advised not to recreate a non-B-tree index.

Rebuilding an Index

Use either of the following two methods to recreate an index:

- Run the **DROP INDEX** statement to delete an index and run the **CREATE INDEX** statement to create an index.

When you delete an index, a temporary exclusive lock is added in the parent table to block related read/write operations. When you create an index, the write operation is locked but the read operation is not. The data is read and scanned by order.

- Run the **REINDEX** statement to recreate an index:
 - When you run the **REINDEX TABLE** statement to recreate an index, an exclusive lock is added to block related read/write operations.
 - When you run the **REINDEX INTERNAL TABLE** statement to recreate an index for a **desc** table (), an exclusive lock is added to block read/write operations on the table.

Procedure

Assume the ordinary index `areaS_idx` exists in the `area_id` column of the imported table `areaS`. Use either of the following two methods to recreate an index:

- Run the **DROP INDEX** statement to delete the index and run the **CREATE INDEX** statement to create an index.

- a. Delete an index.

```
DROP INDEX areaS_idx;  
DROP INDEX
```
- b. Create an index.

```
CREATE INDEX areaS_idx ON areaS (area_id);  
CREATE INDEX
```
- Run the **REINDEX** statement to recreate an index.
 - Run the **REINDEX TABLE** statement to recreate an index.

```
REINDEX TABLE areaS;  
REINDEX
```
 - Run the **REINDEX INTERNAL TABLE** statement to recreate an index for a **desc** table ().

```
REINDEX INTERNAL TABLE areaS;  
REINDEX
```

16.4.12 Automatic Retry upon SQL Statement Execution Errors

With automatic retry (referred to as CN retry), DWS retries an SQL statement when the execution of a statement fails. If an SQL statement sent from the **gsql** client, JDBC driver, or ODBC driver fails to be executed, the CN can automatically identify the error reported during execution and re-deliver the task to retry.

The restrictions of this function are as follows:

- Functionality restrictions:
 - CN retry increases execution success rate but does not guarantee success.
 - CN retry is enabled by default. In this case, the system records logs about temporary tables. If it is disabled, the system will not record the logs. Therefore, do not repeatedly enable and disable CN retry when temporary tables are used. Otherwise, data inconsistency may occur after a CN retry following a primary/standby switchover.
 - CN retry is enabled by default. In this case, the **unlogged** keyword is ignored in the statement for creating unlogged tables and thereby ordinary tables will be created by using this statement. If CN retry is disabled, the system records logs about unlogged tables. Therefore, do not repeatedly enable and disable CN retry when unlogged tables are used. Otherwise, data inconsistency may occur after a CN retry following a primary/standby switchover.
 - When GDS is used to export data, CN retry is supported. The existing mechanism checks for duplicate files and deletes duplicate files during data export. Therefore, you are advised not to repeatedly export data for the same foreign table unless you are sure that files with the same name in the data directory need to be deleted.
- Error type restrictions:
Only the error types in [Table 16-20](#) are supported.
- Statement type restrictions:
Support single-statement CN retry, stored procedures, functions, and anonymous blocks. Statements in transaction blocks are not supported.
- Statement restrictions of a stored procedure:
 - If an error occurs during the execution of a stored procedure containing **EXCEPTION** (including statement block execution and statement execution in **EXCEPTION**), the stored procedure can be retried. If an

internal error occurs, the stored procedure will retry first, but if the error is captured by **EXCEPTION**, the stored procedure cannot be retried.

- Packages that use global variables are not supported.
- **DBMS_JOB** is not supported.
- **UTL_FILE** is not supported.
- If the stored procedure has printed information (such as **dbms_output.put_line** or **raise info**), the printed information will be output repeatedly when retry occurs, and "Notice: Retry triggered, some message may be duplicated. " will be output before the repeated information.
- Cluster status restrictions:
 - Only DNs or GTMs are faulty.
 - The cluster can be recovered before the number of CN retries reaches the allowed maximum (controlled by **max_query_retry_times**). Otherwise, CN retry may fail.
 - CN retry is not supported during scale-out.
- Data import restrictions:
 - The **COPY FROM STDIN** statement is not supported.
 - The **gsql \copy from** metacommand is not supported.
 - **JDBC CopyManager copyIn** is not supported.

Table 16-20 lists the error types supported by CN retry and the corresponding error codes. You can use the GUC parameter **retry_ecode_list** to set the list of error types supported by CN retry. You are not advised to modify this parameter. To modify it, contact the technical support.

Table 16-20 Error types supported by CN retry

| Error Type | Error Code | Remarks |
|---------------------------------|------------|--|
| CONNECTION_RESET_BY_PEER | YY00
1 | TCP communication errors:
Connection reset by peer
(communication between the CN
and DNs) |
| STREAM_CONNECTION_RESET_BY_PEER | YY00
2 | TCP communication errors: Stream
connection reset by peer
(communication between DNs) |
| LOCK_WAIT_TIMEOUT | YY00
3 | Lock wait timeout |
| CONNECTION_TIMED_OUT | YY00
4 | TCP communication errors:
Connection timed out |
| SET_QUERY_ERROR | YY00
5 | Failed to deliver the SET
command: Set query |
| OUT_OF_LOGICAL_MEMORY | YY00
6 | Failed to apply for memory: Out of
logical memory |

| Error Type | Error Code | Remarks |
|-----------------------------------|------------|--|
| SCTP_MEMORY_ALLOC | YY007 | SCTP communication errors: Memory allocate error |
| SCTP_NO_DATA_IN_BUFFER | YY008 | SCTP communication errors: SCTP no data in buffer |
| SCTP_RELEASE_MEMORY_CLOSE | YY009 | SCTP communication errors: Release memory close |
| SCTP_TCP_DISCONNECT | YY010 | SCTP communication errors: TCP disconnect |
| SCTP_DISCONNECT | YY011 | SCTP communication errors: SCTP disconnect |
| SCTP_REMOTE_CLOSE | YY012 | SCTP communication errors: Stream closed by remote |
| SCTP_WAIT_POLL_UNKNOWN | YY013 | Waiting for an unknown poll: SCTP wait poll unknown |
| SNAPSHOT_INVALID | YY014 | Snapshot invalid |
| ERRCODE_CONNECTION_RECEIVE_WRONG | YY015 | Connection receive wrong |
| OUT_OF_MEMORY | 53200 | Out of memory |
| CONNECTION_FAILURE | 08006 | GTM errors: Connection failure |
| CONNECTION_EXCEPTION | 08000 | Failed to communicate with DN's due to connection errors: Connection exception |
| ADMIN_SHUTDOWN | 57P01 | System shutdown by administrators: Admin shutdown |
| STREAM_REMOTE_CLOSE_SOCKET | XX003 | Remote socket disabled: Stream remote close socket |
| ERRCODE_STREAM_DUPLICATE_QUERY_ID | XX009 | Duplicate query id |
| ERRCODE_STREAM_CONCURRENT_UPDATE | YY016 | Stream concurrent update |
| ERRCODE_LLVM_BAD_ALLOC_ERROR | CG003 | Memory allocation error: Allocate error |
| ERRCODE_LLVM_FATAL_ERROR | CG004 | Fatal error |

| Error Type | Error Code | Remarks |
|---|------------|---|
| HashJoin temporary file reading error (ERRCODE_HASHJOIN_TEMP_FILE_ERROR). | F0011 | File error |
| Buffer file reading error (ERRCODE_BUFFER_FILE_ERROR) | F0012 | File reading error |
| Partition number error (ERRCODE_PARTITION_NUM_CHANGED). | 45003 | During scanning on a list partition table, it is found that the number of partitions is different from that in the optimization phase. This problem usually occurs when the queries and ADD/DROP partitions are concurrently executed. (This error is supported only by clusters of version 8.1.3 or later.) |
| Unmatched schema name (ERRCODE_UNMATCH_OBJECT_SCHEMA) | 42P30 | Unmatched schema name |

To enable CN retry, set the following GUC parameters:

- Mandatory GUC parameters (required by both CNs and DN)
max_query_retry_times

CAUTION

If CN retry is enabled, temporary table data is logged. For data consistency, do not switch the enabled/disabled status for CN retry when the temporary tables are being used by sessions.

-
- Optional GUC parameters
cn_send_buffer_size
max_cn_temp_file_size

16.4.13 Query Band Load Identification

Overview

DWS implements load identification and intra-queue priority control based on query_band. It provides more flexible load identification methods and identifies load queues based on job types, application names, and script names. Users can flexibly configure query_band identification queues based on service scenarios. In addition, priority control of job delivery in the queue is implemented. In the future, priority control of resources in the queue will be gradually implemented.

Administrators can configure the queue associated with `query_band` and estimate the memory limit based on service scenarios and job types to implement more flexible load control and resource management and control. If `query_band` is not configured for the service or the user does not associate `query_band` with an action, the queue associated with the user and the priority in the queue is used by default.

Load Behaviors Supported by `query_band`

`query_band` is a session-level GUC parameter. It is a job identifier of the character data type. Its value can be any string. However, for easier differentiation and configuration, `query_band` only identifies key-value pairs. For example:

```
SET query_band='JobName=abc;AppName=test;UserName=user';
```

JobName=abc, **AppName=test**, and **UserName=user** are independent key-value pairs. Specifications of the `query_band` key-value pairs:

- `query_band` is set in key-value pair mode, that is, 'key=value'. Multiple `query_band` key-value pairs can be set in a session. Multiple key-value pairs are separated by semicolons (;). The maximum length of both the **`query_band`** key-value pair and parameter value is 1024 characters.
- The `query_band` key-value pair supports the following valid characters: digits 0 to 9, uppercase letters A to Z, lowercase letters a to z, '.', '-', '_', and '#'.

`query_band` is configured, and identifies load behaviors, using key-value pairs. The supported load behaviors are described in [Table 16-21](#).

Table 16-21 Load behaviors supported by `QUERY_BAND`

| Type | Behavior | Behavior Description |
|--------------------------------|---|---|
| Workload management (workload) | Resource pool (respool) | <code>query_band</code> associated with a resource pool |
| Workload management (workload) | Priority | Priority in the queue |
| Order | Queue (respool)
Currently, this field is invalid and is used for future extension. | <code>query_band</code> query order |

The "Type" is used to classify load behaviors. Different load behaviors may belong to a same type. For example, both "Resource pool" and a "Priority" belong to "Workload management". The "Behavior" indicates a load behavior associated with a `query_band` key-value pair. The "Behavior description" describes a specific load behavior. The "Order" in the "Type" is used to indicate the priority of the

query_band load behavior identification. When a session has multiple query_band key-value pairs, the query_band key-value pair with a smaller order value is preferentially used to identify a load behavior. Each query_band key-value pair can have multiple associated load behaviors, while one load behavior can only have one associated key-value pair. The query_band load behavior is described as follows:

- Resource pool: query_band can be associated with resource pools. During job execution, if a resource pool is associated with query_band, the resource pool is used in preference. Otherwise, the resource pool associated with the user is used.
 - When query_band is associated with a resource pool, an error is reported if the resource pool does not exist, and the association fails.
 - When query_band is associated with a resource pool, the dependency between query_band and the resource pool is recorded.
 - When a resource pool associated with query_band is deleted, a message is displayed indicating that the resource pool fails to be deleted because of the dependency between query_band and the resource pool.
- Intra-queue priority: query_band can be associated with job priorities, including high, medium, and low. Rush is provided as a special priority (green channel). The default priority is medium. In practice, most jobs use the medium priority, low-priority jobs use the low priority, and privileged jobs use the high priority. It is not recommended that a large number of jobs use the high priority. The rush priority is used only in special scenarios and is not recommended in normal cases.

The intra-queue priority is used to implement the queuing priority.

- In the static load management scenario, when the CN concurrency is insufficient, CN global queuing is triggered. The CN global queue is a priority queue.
- In the dynamic load management scenario, if the DN memory is insufficient, CCN global queuing is triggered. The CCN global queue is a priority queue.
- When the resource pool concurrency or memory is insufficient, resource pool queuing is triggered. The resource pool queue is a priority queue.

The preceding priority queues comply with the following scheduling rules:

- Jobs with a higher priority are scheduled first.
 - After all jobs with a high priority are scheduled, jobs with a low priority are scheduled.
 - In dynamic load management scenarios, the CN global queue does not support the query_band priority.
- Order: The identification order of query_bands can be configured. The default order value is -1. Except the default order value, there are no two query_bands with the same order value. The query_band order is verified when being configured. If there are query_bands with the same order value, the order values are recursively increased by 1 until there are no query_bands with the same order value.
 - If a session has multiple query_band key-value pairs, the query_band key-value pair with a smaller order value is used for load identification.

- **0** is the smallest order value, and the default order value **-1** is the largest order value.
- If the query_bands are all of the same order value, the anterior query_band is used for load identification.
- For example, if in **set query_band='b=1;a=3;c=1'; b=1**, the order value of **b=1** is **-1**, **a=3** is **4**, **c=1** is **1**, **c=1** is used as the query_band for load identification. This design enables load administrators to adjust load scheduling.

Application and Configuration of query_band

- The **pg_workload_action** cross-database system catalog is used to store the query_band action and order. For details, see [PG_WORKLOAD_ACTION](#).
- The default action and order are not stored in the **pg_workload_action** system catalog. If a non-default action is set for query_band, the default action is also displayed when actions are queried. The message <query_band information not found> is displayed when the action and order to be queried are the default query_band action.
- The **gs_wlm_set_queryband_action** function sets the query_band sequence. The maximum length of the first parameter, that is, the query_band key value pair, is 63 characters. For the second parameter, it is case insensitive and multiple actions are separated by semicolons (;). **order** is the default parameter and its default value is **-1**. For details, see **gs_wlm_set_queryband_action** in "Resource Management Functions" in *Data Warehouse Service (DWS) SQL Syntax Reference*.
- The **gs_wlm_set_queryband_order** function sets the query_band sequence. The maximum length of the first parameter, that is, a query_band key value pair, is 63 characters. The value of query_band must be greater than or equal to **-1**. Except the default value **-1**, the value of query_band order must be unique. When setting the query_band order, if there are query_bands with the same order value, the original order value is increased by 1. For details, see **gs_wlm_set_queryband_order** function in section "Resource Management Functions" in *Data Warehouse Service (DWS) SQL Syntax Reference*.
- You can use the **gs_wlm_get_queryband_action** function to query the **query_band** action. For details, see **gs_wlm_set_queryband_action** in "Resource Management Functions" in *Data Warehouse Service (DWS) SQL Syntax Reference*.
- **pg_queryband_action** provides the system view for querying all **query_band** actions. For details, see [PG_QUERYBAND_ACTION](#).
- The query_band priority is displayed as an integer in the load management view ([PG_SESSION_WLMSTAT](#)). The mapping between numbers and priorities is as follows:
 - 0: not controlled by load management
 - 1: low
 - 2: medium
 - 4: high
 - 8: rush
- Permission control: Except initial users, other users have the permission to set and query query_band only when they are authorized.

 NOTE

- When all running jobs are canceled in batches or the maximum number of concurrent jobs in a queue is 1 and only one queue is running jobs, the CN may be triggered to automatically wake up jobs. As a result, jobs are not delivered by priority.

Examples

Step 1 Set the associated resource pool to **p1**, priority to **rush**, and order to **1** for query_band **JobName** to **abc**.

```
SELECT * FROM gs_wlm_set_queryband_action('JobName=abc','respool=p1;priority=rush',1);
gs_wlm_set_queryband_action
-----
t
(1 row)
```

Step 2 Change the associated resource pool to **p2** for query_band **JobName=abc**.

```
SELECT * FROM gs_wlm_set_queryband_action('JobName=abc','respool=p2');
gs_wlm_set_queryband_action
-----
t
(1 row)
```

Step 3 Change the priority to **high** for query_band **JobName=abc**.

```
SELECT * FROM gs_wlm_set_queryband_action('JobName=abc','priority=high');
gs_wlm_set_queryband_action
-----
t
(1 row)
```

Step 4 Change the order to **3** for query_band **JobName=abc**.

```
SELECT * FROM gs_wlm_set_queryband_order('JobName=abc',3);
gs_wlm_set_queryband_order
-----
t
(1 row)
```

Step 5 Query the load behaviors associated with query_band.

```
SELECT * FROM pg_queryband_action;
qband | respool_id | respool | priority | qborder
-----+-----+-----+-----+-----
JobName=abc | 17119 | p2 | high | 1
(1 row)
```

Step 6 In **query_band**, set the priority of **AppName=test** to **Low**, associate the user with the resource pool, and use the default sequence.

```
SELECT * FROM gs_wlm_set_queryband_action('AppName=test','priority=low');
gs_wlm_set_queryband_action
-----
t
(1 row)
```

Step 7 Query the load behaviors associated with query_band.

```
SELECT * FROM pg_queryband_action;
qband | respool_id | respool | priority | qborder
-----+-----+-----+-----+-----
AppName=test | 0 | NULL | low | -1
JobName=abc | 16754 | p2 | high | 3
(2 rows)
```

Step 8 In **query_band**, cancel all the workload behaviors associated with **JobName=abc** and set them to default behaviors.

```
SELECT * FROM gs_wlm_set_queryband_action('JobName=abc','respool=null;priority=medium',-1);
NOTICE: The respool of query_band(JobName=abc) will be removed.
```

```
NOTICE: The priority of query_band(JobName=abc) will be removed.
gs_wlm_set_queryband_action
-----
t
(1 row)
```

Step 9 Query the load behaviors associated with query_band.

```
SELECT * FROM pg_queryband_action;
  qband   | respool_id | respool | priority | qborder
-----+-----+-----+-----+-----
AppName=test |      0 | NULL   | low      |      -1
(1 row)

----End
```

16.5 SQL Tuning Examples

16.5.1 Case: Selecting an Appropriate Distribution Column

Distribution columns are used to distribute data to different nodes. A proper distribution key can avoid data skew.

When performing join query, you are advised to select the join condition in the query as the distribution key. When a join condition is used as a distribution key, related data is distributed locally on DNs, reducing the cost of data flow between DNs and improving the query speed.

Before optimization

Use **a** as the distribution column of **t1** and **t2**. The table definition is as follows:

```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH (a);
CREATE TABLE t2 (a int, b int) DISTRIBUTE BY HASH (a);
```

The following query is executed:

```
SELECT * FROM t1, t2 WHERE t1.a = t2.b;
```

In this case, the execution plan contains **Streaming(type: REDISTRIBUTE)**, that is, the DN redistributes data to all DNs based on the selected column. This will cause a large amount of data to be transmitted between DNs, as shown in [Figure 16-19](#).

Figure 16-19 Selecting an appropriate distribution column (1)

EXPLAIN PERFORMANCE SELECT * FROM t1, t2 WHERE t1.a = t2.b;

| QUERY PLAN | | | | | | | | | | |
|------------|-----------------------------------|----------------|--------|--------|------------|----------------|----------|---------|---------|---------|
| id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs |
| 1 | -> Streaming (type: GATHER) | 0.760 | 0 | 30 | | 24KB | | | 16 | 37.96 |
| 2 | -> Hash Join (3,5) | 0.295, 0.428 | 0 | 30 | | [8KB, 8KB] | 1MB | | 16 | 29.96 |
| 3 | -> Streaming (type: REDISTRIBUTE) | [0, 0] | 0 | 30 | 10 | [0, 0] | 2MB | | 8 | 15.49 |
| 4 | -> Seq Scan on dbadmin.t2 | [0.001, 0.002] | 0 | 30 | | [32KB, 32KB] | 1MB | | 8 | 14.14 |
| 5 | -> Hash | [0.003, 0.003] | 0 | 29 | 14 | [264KB, 264KB] | 16MB | | 8 | 14.14 |
| 6 | -> Seq Scan on dbadmin.t1 | [0.001, 0.002] | 0 | 30 | | [32KB, 32KB] | 1MB | | 8 | 14.14 |

Predicate Information (identified by plan id)

```
2 --Hash Join (3,5)
   Hash Cond: (t2.b = t1.a)
```

After optimization

Use the join condition in the query as the distribution key and run the following statement to change the distribution key of **t2** as **b**:

```
ALTER TABLE t2 DISTRIBUTE BY HASH (b);
```

After the distribution column of table **t2** is changed to column **b**, the execution plan does not contain **Streaming(type: REDISTRIBUTE)**. This reduces the amount of communication data between DN and reduces the execution time from 8.7 ms to 2.7 ms, improving query performance, as shown in [Figure 16-20](#).

Figure 16-20 Selecting an appropriate distribution column (2)

pg_get_tabledef

```
EXPLAIN PERFORMANCE SELECT * FROM t1, t2 WHERE t1.a = t2.b;
```

| id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs |
|----|-----------------------------|----------------|--------|--------|------------|--------------|----------|---------|---------|---------|
| 1 | -> Streaming (type: GATHER) | 2.727 | 0 | 30 | | 24KB | | | 16 | 36.59 |
| 2 | -> Hash Join (3,4) | [0.006, 0.007] | 0 | 30 | | [8KB, 8KB] | 1MB | | 16 | 28.59 |
| 3 | -> Seq Scan on dbadmin.t1 | [0.001, 0.002] | 0 | 30 | 14 | [16KB, 16KB] | 1MB | | 8 | 14.14 |
| 4 | -> Hash | [0, 0] | 0 | 29 | 14 | [0, 0] | 16MB | | 8 | 14.14 |
| 5 | -> Seq Scan on dbadmin.t2 | [0, 0] | 0 | 30 | | [0, 0] | 1MB | | 8 | 14.14 |

Predicate Information (identified by plan id)

```
2 --Hash Join (3,4)
Hash Cond: (t1.a = t2.b)
```

16.5.2 Case: Creating an Appropriate Index

Creating a proper index can accelerate the retrieval of data rows in a table. Indexes occupy disk space and reduce the speed of adding, deleting, and updating rows. If data needs to be updated very frequently or disk space is limited, you need to limit the number of indexes. Create indexes for large tables. Because the more data in the table, the more effective the index is. You are advised to create indexes on:

- Columns that need to be queried frequently
- Joined columns. For a query on joined columns, you are advised to create a composite index on the joined columns. For example, if the join condition is **select * from t1 join t2 on t1.a=t2.a and t1.b=t2.b**. You can create a composite index on the **a** and **b** columns of table **t1**.
- Columns having filter criteria (especially scope criteria) of a **where** clause
- Columns that appear after **order by**, **group by**, and **distinct**

Before optimization

The column-store partitioned table **orders** is defined as follows:

```
pg_get_tabledef
-----
SET search_path = dbadmin;
CREATE TABLE orders (
  o_orderkey bigint NOT NULL,
  o_custkey bigint NOT NULL,
  o_orderstatus character(1) NOT NULL,
  o_totalprice numeric(15,2) NOT NULL,
  o_orderdate timestamp(0) without time zone NOT NULL,
  o_orderpriority character(15) NOT NULL,
  o_clerk character(15) NOT NULL,
  o_shippriority bigint NOT NULL,
  o_comment character varying(79) NOT NULL
)
WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)
DISTRIBUTE BY HASH(o_orderkey)
TO GROUP group_version1
PARTITION BY RANGE (o_orderdate)
(
  PARTITION o_orderdate_1 VALUES LESS THAN ('1993-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_2 VALUES LESS THAN ('1994-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_3 VALUES LESS THAN ('1995-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_4 VALUES LESS THAN ('1996-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_5 VALUES LESS THAN ('1997-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_6 VALUES LESS THAN ('1998-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_7 VALUES LESS THAN ('1999-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
)
ENABLE ROW MOVEMENT;
(1 row)
```

Run the SQL statement to query the execution plan when no index is created. It is found that the execution time is 48 milliseconds.

```
EXPLAIN PERFORMANCE SELECT * FROM orders WHERE o_custkey = '1106459';
```

| QUERY PLAN | | | | | | | | | | | |
|------------|---|------------------|--------|--------|------------|--------------|----------|---------|---------|----------|--|
| id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs | |
| 1 | -> Row Adapter | 48.588 | 6 | 16 | | 82KB | | | 123 | 94931.88 | |
| 2 | -> Vector Streaming (type: GATHER) | 48.491 | 6 | 16 | | 249KB | | | 123 | 94931.88 | |
| 3 | -> Vector Partition Iterator | [45.479, 45.479] | 6 | 16 | | [17KB, 17KB] | 1MB | | 123 | 94923.88 | |
| 4 | -> Partitioned CStore Scan on public.orders | [45.157, 45.157] | 6 | 16 | | [1MB, 1MB] | 1MB | | 123 | 94923.88 | |

After optimization

The filtering condition column of the **where** clause is **o_custkey**. Add an index to the **o_custkey** column.

```
CREATE INDEX idx_o_custkey ON orders (o_custkey) LOCAL;
```

Run the SQL statement to query the execution plan after the index is created. It is found that the execution time is 18 milliseconds.

| QUERY PLAN | | | | | | | | | | | |
|------------|---|------------------|--------|--------|------------|----------------|----------|---------|---------|---------|--|
| id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs | |
| 1 | -> Row Adapter | 18.889 | 6 | 16 | | 82KB | | | 123 | 18 | |
| 2 | -> Vector Streaming (type: GATHER) | 18.881 | 6 | 16 | | 249KB | | | 123 | 18 | |
| 3 | -> Vector Partition Iterator | [12.224, 12.224] | 6 | 16 | | [271KB, 271KB] | 1MB | | 123 | 18 | |
| 4 | -> Partitioned CStore Index Scan using idx_o_custkey on public.orders | [18.695, 18.695] | 6 | 16 | | [1MB, 1MB] | 1MB | | 123 | 18 | |

16.5.3 Case: Adding NOT NULL for JOIN Columns

If there are many **NULL** values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.

Before optimization

```
SELECT
*
FROM
( ( SELECT
STARTTIME STTIME,
SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,
SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,
SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,
SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,
SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,
SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,
SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL
FROM
PS.SDR_WEB_BSCRNC_1DAY SDR
INNER JOIN (SELECT
BSCRNC_ID,
BSCRNC_NAME,
ACCESS_TYPE,
ACCESS_TYPE_ID
FROM
nethouse.DIM_LOC_BSCRNC
GROUP BY
BSCRNC_ID,
BSCRNC_NAME,
ACCESS_TYPE,
ACCESS_TYPE_ID) DIM
ON SDR.BSCRNC_ID = DIM.BSCRNC_ID
AND DIM.ACCESS_TYPE_ID IN (0,1,2)
INNER JOIN nethouse.DIM_RAT_MAPPING RAT
ON (RAT.RAT = SDR.RAT)
WHERE
( ( STARTTIME >= 1461340800
```

```
AND STARTTIME < 1461427200 )  
AND RAT.ACCESS_TYPE_ID IN (0,1,2)  
GROUP BY STTIME ) ) ;
```

Figure 16-21 shows the execution plan.

Figure 16-21 Adding NOT NULL for JOIN columns (1)

| id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs |
|----|---|--------------------|-----------|---------|------------------|----------|---------|----------|--------------------|
| 1 | Row Adapter | 1805.792 | 1 | 72 | 72KB | 1 | | 160 | 1.00444120.99 |
| 2 | Vector Streaming (type: GATHER) | 1805.779 | 1 | 72 | 144KB | 1 | | 160 | 1.00444120.99 |
| 3 | Vector Hash Aggregate | 15621.425,3479.484 | 1 | 1 | 13004KB, 9000KB | 1 | 16KB | | 55 1.00444120.99 |
| 4 | Vector Streaming (type: REDISTRIBUTE) | 15621.305,3479.103 | 72 | 2 | 1444KB, 1440KB | 1 | 1MB | | 55 1.00444120.99 |
| 5 | Vector Hash Aggregate | 13914.657,3434.515 | 72 | 2 | 13013KB, 9010KB | 1 | 16KB | [75, 78] | 55 1.00444120.99 |
| 6 | Vector Hash Join (1,1) | 13234.674,3545.293 | 3445920 | 2094077 | 17784KB, 11111KB | 1 | 16KB | | 55 1.00444120.99 |
| 7 | Vector Hash Aggregate | 15.430,1.473 | 1087848 | 15109 | 1310KB, 1310KB | 1 | 16KB | [48, 48] | 32 1.074.99 |
| 8 | Cost-based Scan on dim_loc_bscrcnc | 11.071,1.229 | 1087848 | 15109 | 1412KB, 1412KB | 1 | 1MB | | 32 1.072.77 |
| 9 | Vector Hash Join (1,2) | 12441.330,2753.418 | 143194616 | 1257925 | 1710KB, 1310KB | 1 | 16KB | | 60 1.017949.99 |
| 10 | Cost-based Scan on sdr_web_bscrcnc_1day sdr | 1485.201,2751.045 | 143194616 | 1257925 | 1310KB, 1310KB | 1 | 1MB | | 64 1.019324.25 |
| 11 | Cost-based Scan on dim_rat_mapping rat | 0.070,0.111 | 288 | 4 | 577KB, 577KB | 1 | 1MB | [16, 16] | 8 1.00.05 |

After optimization

- As shown in Figure 16-21, the sequential scan phase is time consuming.
- The JOIN performance is poor because a large number of null values exist in the JOIN column **BSCRNC_ID** of the PS.SDR_WEB_BSCRNC_1DAY table.

Therefore, you are advised to manually add **NOT NULL** for **JOIN** columns in the statement, as shown below:

```
SELECT  
*  
FROM  
( ( SELECT  
STARTTIME STTIME,  
SUM(NVL(PAGE_DELAY_MSEL,0)) PAGE_DELAY_MSEL,  
SUM(NVL(PAGE_SUCCEED_TIMES,0)) PAGE_SUCCEED_TIMES,  
SUM(NVL(FST_PAGE_REQ_NUM,0)) FST_PAGE_REQ_NUM,  
SUM(NVL(PAGE_AVG_SIZE,0)) PAGE_AVG_SIZE,  
SUM(NVL(FST_PAGE_ACK_NUM,0)) FST_PAGE_ACK_NUM,  
SUM(NVL(DATATRANS_DW_DURATION,0)) DATATRANS_DW_DURATION,  
SUM(NVL(PAGE_SR_DELAY_MSEL,0)) PAGE_SR_DELAY_MSEL  
FROM  
PS.SDR_WEB_BSCRNC_1DAY SDR  
INNER JOIN (SELECT  
BSCRNC_ID,  
BSCRNC_NAME,  
ACCESS_TYPE,  
ACCESS_TYPE_ID  
FROM  
nethouse.DIM_LOC_BSCRNC  
GROUP BY  
BSCRNC_ID,  
BSCRNC_NAME,  
ACCESS_TYPE,  
ACCESS_TYPE_ID) DIM  
ON SDR.BSCRNC_ID = DIM.BSCRNC_ID  
AND DIM.ACCESS_TYPE_ID IN (0,1,2)  
INNER JOIN nethouse.DIM_RAT_MAPPING RAT  
ON (RAT.RAT = SDR.RAT)  
WHERE  
( ( STARTTIME >= 1461340800  
AND STARTTIME < 1461427200 )  
AND RAT.ACCESS_TYPE_ID IN (0,1,2)  
and SDR.BSCRNC_ID is not null  
GROUP BY  
STTIME ) ) A;
```

Figure 16-22 shows the execution plan.

Figure 16-22 Adding NOT NULL for JOIN columns (2)

| id | operation | A-time | A-rows | E-rows | Peak Memory | Memory | A-width | E-width | E-costs |
|----|--|--------------------|---------|--------|------------------|--------|----------|---------|--------------------|
| 1 | -> Row Adapter | 873.795 | 1 | 72 | 72KB | | | | 140 121433605.45 |
| 2 | -> Vector Streaming (type: GATHER) | 873.784 | 1 | 72 | 44KB | | | | 140 121433605.45 |
| 3 | -> Vector Hash Aggregate | [685.940, 744.454] | 1 | 1 | [3004KB, 3005KB] | 1MB | [75, 78] | | 55 14686577.84 |
| 4 | -> Vector Streaming (type: REDISTRIBUTE) | [685.810, 744.365] | 72 | 1 | [2404KB, 2405KB] | 1MB | | | 55 14686577.83 |
| 5 | -> Vector Hash Aggregate | [590.319, 710.812] | 72 | 1 | [3015KB, 3016KB] | 1MB | [75, 78] | | 55 14686577.84 |
| 6 | -> Vector Hash Join (7,10) | [561.469, 661.631] | 3665920 | 102203 | [2769KB, 2769KB] | 1MB | | | 55 14686533.77 |
| 7 | -> Vector Hash Join (7,9) | [545.846, 636.404] | 3686400 | 44859 | [2338KB, 2338KB] | 1MB | | | 40 1596787.26 |
| 8 | -> CStore Scan on sdr_web_becno_iday sdr | [541.494, 628.405] | 3686400 | 78509 | [518KB, 535KB] | 1MB | | | 44 1595524.20 |
| 9 | -> CStore Scan on sdr_rat_mapping rat | [0.031, 0.107] | 288 | 4 | [577KB, 577KB] | 1MB | [16, 16] | | 4 150.03 |
| 10 | -> Vector Subquery Scan on dim | [5.526, 6.960] | 1087848 | 15109 | [40KB, 40KB] | 1MB | [19, 19] | | 7 1726.04 |
| 11 | -> Vector Hash Aggregate | [5.497, 6.931] | 1087848 | 15109 | [5339KB, 2550KB] | 1MB | [48, 48] | | 32 1574.95 |
| 12 | -> CStore Scan on dim_loo_becno | [1.087, 1.424] | 1087848 | 15109 | [1412KB, 1412KB] | 1MB | | | 32 1272.77 |

16.5.4 Case: Configuring cost_param for Better Query Performance

The `cost_param` parameter is used to control use of different estimation methods in specific customer scenarios, allowing estimated values to be close to onsite values. This parameter can control various methods simultaneously by performing AND (&) operations on the bit for each method. A method is selected if its value is not 0.

Scenario 1: Before Optimization

If **bit0** of **cost_param** is set to **1**, an improved mechanism is used for estimating the selection rate of non-equi-joins. This method is more accurate for estimating the selection rate of joins between two identical tables. The following example describes the optimization scenario when **bit0** of **cost_param** is set to **1**. In V300R002C00 and later, **cost_param & 1=0** is not used. That is, an optimized formula is selected for calculation.

NOTE

The selection rate indicates the percentage for which the number of rows meeting the join conditions account of the **JOIN** results when the **JOIN** relationship is established between two tables.

The table structure is as follows:

```
CREATE TABLE LINEITEM
(
  L_ORDERKEY BIGINT NOT NULL
, L_PARTKEY BIGINT NOT NULL
, L_SUPPKEY BIGINT NOT NULL
, L_LINENUMBER BIGINT NOT NULL
, L_QUANTITY DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL
, L_DISCOUNT DECIMAL(15,2) NOT NULL
, L_TAX DECIMAL(15,2) NOT NULL
, L_RETURNFLAG CHAR(1) NOT NULL
, L_LINESTATUS CHAR(1) NOT NULL
, L_SHIPDATE DATE NOT NULL
, L_COMMITDATE DATE NOT NULL
, L_RECEIPTDATE DATE NOT NULL
, L_SHIPINSTRUCT CHAR(25) NOT NULL
, L_SHIPMODE CHAR(10) NOT NULL
, L_COMMENT VARCHAR(44) NOT NULL
) with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(L_ORDERKEY);

CREATE TABLE ORDERS
(
  O_ORDERKEY BIGINT NOT NULL
, O_CUSTKEY BIGINT NOT NULL
, O_ORDERSTATUS CHAR(1) NOT NULL
, O_TOTALPRICE DECIMAL(15,2) NOT NULL
```

```
, O_ORDERDATE DATE NOT NULL
, O_ORDERPRIORITY CHAR(15) NOT NULL
, O_CLERK CHAR(15) NOT NULL
, O_SHIPPRIORITY BIGINT NOT NULL
, O_COMMENT VARCHAR(79) NOT NULL
)with (orientation = column, COMPRESSION = MIDDLE) distribute by hash(O_ORDERKEY);
```

The query statements are as follows:

```
explain verbose select
count(*) as numwait
from
lineitem l1,
orders
where
o_orderkey = l1.l_orderkey
and o_orderstatus = 'F'
and l1.l_receiptdate > l1.l_commitdate
and not exists (
select
*
from
lineitem l3
where
l3.l_orderkey = l1.l_orderkey
and l3.l_suppkey <> l1.l_suppkey
and l3.l_receiptdate > l3.l_commitdate
)
order by
numwait desc;
```

The following figure shows the execution plan. (When **verbose** is used, **distinct** is added for column selection which is controlled by **cost off/on**. The hash join rows show the estimated number of distinct values and the other rows do not.)

| id | operation | E-rows | E-distinct | E-width | E-costs |
|----|--------------------------------------|--------|------------|---------|---------|
| 1 | -> Row Adapter | 1 | | 8 | 39.36 |
| 2 | -> Vector Sort | 1 | | 8 | 39.36 |
| 3 | -> Vector Aggregate | 1 | | 8 | 39.34 |
| 4 | -> Vector Streaming (type: GATHER) | 2 | | 8 | 39.34 |
| 5 | -> Vector Aggregate | 2 | | 8 | 39.25 |
| 6 | -> Vector Hash Anti Join (7, 10) | 2 | 4, 5 | 0 | 39.24 |
| 7 | -> Vector Hash Join (8,9) | 2 | 200, 1 | 16 | 26.12 |
| 8 | -> CStore Scan on public.lineitem l1 | 7 | | 16 | 13.05 |
| 9 | -> CStore Scan on public.orders | 1 | | 8 | 13.05 |
| 10 | -> CStore Scan on public.lineitem l3 | 7 | | 16 | 13.05 |

Scenario 1: After Optimization

These queries are from Anti Join connected in the **lineitem** table. When **cost_param & bit0** is **0**, the estimated number of Anti Join rows greatly differs from that of the actual number of rows, compromising the query performance. You can estimate the number of Anti Join rows more accurately by setting **cost_param & bit0** to **1** to improve the query performance. The optimized execution plan is as follows:

| id | operation | E-rows | E-memory | E-width | E-costs |
|----|--|------------|----------|---------|--------------|
| 1 | -> Row Adapter | 1 | | 0 | 9104892.37 9 |
| 2 | -> Vector Sort | 1 | | 0 | 9104892.37 9 |
| 3 | -> Vector Aggregate | 1 | | 0 | 9104892.35 8 |
| 4 | -> Vector Streaming (type: GATHER) | 48 | | 0 | 9104892.35 8 |
| 5 | -> Vector Aggregate | 48 | 1MB | 0 | 9104890.82 5 |
| 6 | -> Vector Hash Join (7.12) | 2526630903 | 929MB | 0 | 8973295.45 4 |
| 7 | -> Vector Hash Anti Join (8. 10) | 1999996587 | 3178MB | 8 | 7198231.14 |
| 8 | -> Vector Partition Iterator | 1999996587 | 1MB | 16 | 3000158.25 |
| 9 | -> Partitioned CStore Scan on public.lineitem 11 | 1999996587 | 1MB | 16 | 3000158.25 1 |
| 10 | -> Vector Partition Iterator | 1999996587 | 1MB | 16 | 3000158.25 |
| 11 | -> Partitioned CStore Scan on public.lineitem 13 | 1999996587 | 1MB | 16 | 3000158.25 |
| 12 | -> Vector Partition Iterator | 730839014 | 1MB | 8 | 589611.00 |
| 13 | -> Partitioned CStore Scan on public.orders | 730839014 | 1MB | 8 | 589611.00 |

Scenario 2: Before Optimization

If **bit1** is set to **1** (**set cost_param=2**), the selection rate is estimated based on multiple filter criteria. The lowest selection rate among all filter criteria, but not the product of the selection rates for two tables under a specific filter criterion, is used as the total selection rate. This method is more accurate when a close correlation exists between the columns to be filtered. The following example describes the optimization scenario when **bit1** of **cost_param** is set to **1**.

The table structure is as follows:

```
CREATE TABLE NATION
(
  N_NATIONKEY INT NOT NULL
, N_NAME CHAR(25) NOT NULL
, N_REGIONKEY INT NOT NULL
, N_COMMENT VARCHAR(152)
) distribute by replication;
CREATE TABLE SUPPLIER
(
  S_SUPPKEY BIGINT NOT NULL
, S_NAME CHAR(25) NOT NULL
, S_ADDRESS VARCHAR(40) NOT NULL
, S_NATIONKEY INT NOT NULL
, S_PHONE CHAR(15) NOT NULL
, S_ACCTBAL DECIMAL(15,2) NOT NULL
, S_COMMENT VARCHAR(101) NOT NULL
) distribute by hash(S_SUPPKEY);
CREATE TABLE PARTSUPP
(
  PS_PARTKEY BIGINT NOT NULL
, PS_SUPPKEY BIGINT NOT NULL
, PS_AVAILQTY BIGINT NOT NULL
, PS_SUPPLYCOST DECIMAL(15,2) NOT NULL
, PS_COMMENT VARCHAR(199) NOT NULL
) distribute by hash(PS_PARTKEY);
```

The query statements are as follows:

```
set cost_param=2;
explain verbose select
nation,
sum(amount) as sum_profit
from
(
  select
  n_name as nation,
  l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
  from
  supplier,
  lineitem,
```

```

partsupp,
nation
where
s_suppkey = l_suppkey
and ps_suppkey = l_suppkey
and ps_partkey = l_partkey
and s_nationkey = n_nationkey
) as profit
group by nation
order by nation;

```

When **bit1** of **cost_param** is **0**, the execution plan is shown as follows:

| id | operation | E-rows | E-distinct | E-width | E-costs |
|----|-----------------------------------|--------|------------|---------|---------|
| 1 | -> Sort | 1 | | 208 | 61.52 |
| 2 | -> HashAggregate | 1 | | 208 | 61.51 |
| 3 | -> Streaming (type: GATHER) | 2 | | 208 | 61.51 |
| 4 | -> HashAggregate | 2 | | 208 | 61.36 |
| 5 | -> Hash Join (6,7) | 2 | 20, 15 | 176 | 61.33 |
| 6 | -> Seq Scan on public.nation | 40 | | 108 | 20.20 |
| 7 | -> Hash | 2 | | 76 | 41.04 |
| 8 | -> Hash Join (9,16) | 2 | 10, 13 | 76 | 41.04 |
| 9 | -> Streaming(type: REDISTRIBUTE) | 2 | | 88 | 27.73 |
| 10 | -> Hash Join (11,14) | 2 | 10, 13 | 88 | 27.62 |
| 11 | -> Streaming(type: REDISTRIBUTE) | 20 | | 70 | 14.19 |
| 12 | -> Row Adapter | 21 | | 70 | 13.01 |
| 13 | -> CStore Scan on public.lineitem | 20 | | 70 | 13.01 |
| 14 | -> Hash | 21 | | 34 | 13.13 |
| 15 | -> Seq Scan on public.partsupp | 20 | | 34 | 13.13 |
| 16 | -> Hash | 21 | | 12 | 13.13 |
| 17 | -> Seq Scan on public.supplier | 20 | | 12 | 13.13 |

Scenario 2: After Optimization

In the preceding queries, the hash join criteria of the supplier, lineitem, and partsupp tables are setting **lineitem.l_suppkey** to **supplier.s_suppkey** and **lineitem.l_partkey** to **partsupp.ps_partkey**. Two filter criteria exist in the hash join conditions. **lineitem.l_suppkey** in the first filter criteria and **lineitem.l_partkey** in the second filter criteria are two columns with strong relationship of the lineitem table. In this situation, when you estimate the rate of the hash join conditions, if **cost_param & bit1** is **0**, the selection rate is estimated based on multiple filter criteria. The lowest selection rate among all filter criteria, but not the product of the selection rates for two tables under a specific filter criterion, is used as the total selection rate. This method is more accurate when a close correlation exists between the columns to be filtered. The plan after optimization is shown as follows:

| id | operation | E-rows | E-distinct | E-width | E-costs |
|----|-----------------------------------|--------|------------|---------|---------|
| 1 | -> Sort | 10 | | 208 | 64.42 |
| 2 | -> HashAggregate | 10 | | 208 | 64.23 |
| 3 | -> Streaming (type: GATHER) | 20 | | 208 | 64.23 |
| 4 | -> HashAggregate | 20 | | 208 | 62.71 |
| 5 | -> Hash Join (6,7) | 20 | 20, 10 | 176 | 62.46 |
| 6 | -> Seq Scan on public.nation | 40 | | 108 | 20.20 |
| 7 | -> Hash | 20 | | 76 | 41.97 |
| 8 | -> Hash Join (9,16) | 20 | 10, 13 | 76 | 41.97 |
| 9 | -> Streaming(type: REDISTRIBUTE) | 20 | | 82 | 28.54 |
| 10 | -> Hash Join (11,14) | 20 | 10, 13 | 82 | 27.63 |
| 11 | -> Streaming(type: REDISTRIBUTE) | 20 | | 70 | 14.19 |
| 12 | -> Row Adapter | 21 | | 70 | 13.01 |
| 13 | -> CStore Scan on public.lineitem | 20 | | 70 | 13.01 |
| 14 | -> Hash | 21 | | 12 | 13.13 |
| 15 | -> Seq Scan on public.supplier | 20 | | 12 | 13.13 |
| 16 | -> Hash | 21 | | 34 | 13.13 |
| 17 | -> Seq Scan on public.partsupp | 20 | | 34 | 13.13 |

16.5.5 Case: Adjusting the Partial Clustering Key

Partial Cluster Key (PCK) is an index technology that uses min/max indexes to quickly scan base tables in column storage. Partial cluster key can specify multiple

columns, but you are advised to specify no more than two columns. It can be used to accelerated queries on large column-store tables.

Before Optimization

Create a column-store table **orders_no_pck** without partial clustering (PCK). The table is defined as follows:

```
pg_get_tabledef
-----
SET search_path = dbadmin;
CREATE TABLE orders_no_pck (
  o_orderkey bigint NOT NULL,
  o_custkey bigint NOT NULL,
  o_orderstatus character(1) NOT NULL,
  o_totalprice numeric(15,2) NOT NULL,
  o_orderdate timestamp(0) without time zone NOT NULL,
  o_orderpriority character(15) NOT NULL,
  o_clerk character(15) NOT NULL,
  o_shippriority bigint NOT NULL,
  o_comment character varying(79) NOT NULL
)
WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)
DISTRIBUTE BY HASH(o_orderkey)
TO GROUP group_version1;
(1 row)
```

Run the following SQL statement to query the execution plan of a point query:

```
EXPLAIN PERFORMANCE
SELECT * FROM orders_no_pck
WHERE o_orderkey = '13095143'
ORDER BY o_orderdate;
```

As shown in the following figure, the execution time is 48 ms. Check **Datanode Information**. It is found that the filter time is 19 ms and the CUNone ratio is 0.

```
gaussdb> EXPLAIN PERFORMANCE
gaussdb> SELECT * FROM orders_no_pck
gaussdb> WHERE o_orderkey = '13095143'
gaussdb> ORDER BY o_orderdate;

QUERY PLAN
-----
id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs
---|---|---|---|---|---|---|---|---|---|---
1 | --Row Adapter | 48.182 | 1 | 3 | 1 | 82KB | 1 | 123 | 94838.01
2 | --> Vector Streaming (type: GATHER) | 48.175 | 1 | 3 | 1 | 82KB | 1 | 123 | 94838.01
3 | --> Vector Sort | [44.268, 44.772] | 1 | 3 | 1 | [338KB, 411KB] | 10MB | 123 | 94838.01
4 | --> CStore Scan on public.orders_no_pck | [44.157, 44.669] | 1 | 1 | 1 | [1MB, 1MB] | 1MB | 123 | 94838.00

Predicate Information (identified by plan id)
```

```
Datanode Information (identified by plan id)
-----
1 --Row Adapter
  (actual time=48.181..48.182 rows=1 loops=1)
  (CPU: ex c/r=676, ex row=1, ex c/c=676, inc c/c=4818100)
2 --Vector Streaming (type: GATHER)
  (actual time=48.174..48.175 rows=1 loops=1)
  (Buffers: 0)
  (CPU: ex c/r=4817424, ex row=1, ex c/c=4817424, inc c/c=4817424)
3 --Vector Sort
  dn_6001_6002 (actual time=44.461..44.461 rows=0 loops=1)
  dn_6003_6004 (actual time=44.259..44.268 rows=1 loops=1)
  dn_6005_6006 (actual time=44.772..44.772 rows=0 loops=1)
  dn_6001_6002 (Buffers: shared hit=389)
  dn_6003_6004 (Buffers: shared hit=389)
  dn_6005_6006 (Buffers: shared hit=389)
  dn_6001_6002 (CPU: ex c/r=0, ex row=0, ex c/c=11343, inc c/c=4446062)
  dn_6003_6004 (CPU: ex c/r=10101, ex row=1, ex c/c=10101, inc c/c=4425810)
  dn_6005_6006 (CPU: ex c/r=0, ex row=0, ex c/c=10257, inc c/c=4477201)
4 --CStore Scan on public.orders_no_pck
  dn_6001_6002 (actual time=44.348..44.348 rows=0 loops=1) (filter time=19.721) (RoughCheck CU: CUNone: 0, CUSome: 84)
  dn_6003_6004 (actual time=38.784..44.157 rows=1 loops=1) (filter time=19.739) (RoughCheck CU: CUNone: 0, CUSome: 84)
  dn_6005_6006 (actual time=44.669..44.669 rows=0 loops=1) (filter time=19.568) (RoughCheck CU: CUNone: 0, CUSome: 84)
  dn_6001_6002 (Buffers: shared hit=389)
  dn_6003_6004 (Buffers: shared hit=389)
  dn_6005_6006 (Buffers: shared hit=389)
  dn_6001_6002 (CPU: ex c/r=0, ex row=5007635, ex c/c=4434719, inc c/c=4434719)
  dn_6003_6004 (CPU: ex c/r=0, ex row=5002075, ex c/c=4415709, inc c/c=4415709)
  dn_6005_6006 (CPU: ex c/r=0, ex row=4989390, ex c/c=4466944, inc c/c=4466944)
```

After Optimization

The created column-store table **orders_pck** is defined as follows:

```
pg_get_tabledef
-----
SET search_path = dbadmin;
CREATE TABLE orders_pck (
  o_orderkey bigint NOT NULL,
  o_custkey bigint NOT NULL,
  o_orderstatus character(1) NOT NULL,
  o_totalprice numeric(15,2) NOT NULL,
  o_orderdate timestamp(0) without time zone NOT NULL,
  o_orderpriority character(15) NOT NULL,
  o_clerk character(15) NOT NULL,
  o_shippriority bigint NOT NULL,
  o_comment character varying(79) NOT NULL
)
WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)
DISTRIBUTE BY HASH(o_orderkey)
TO GROUP group_version1;
(1 row)
```

Use **ALTER TABLE** to set the **o_orderkey** field to **PCK**:

```
postgres=> ALTER TABLE orders_pck ADD PARTIAL CLUSTER KEY(o_orderkey);
ALTER TABLE
```

Run the following SQL statement to query the execution plan of the same point query SQL statement again:

```
EXPLAIN PERFORMANCE
SELECT * FROM orders_pck
WHERE o_orderkey = '13095143'
ORDER BY o_orderdate;
```

As shown in the following figure, the execution time is 5 ms. Check **Datanode Information**. It is found that the filter time is 0.5 ms and the CUNone ratio is 82. The higher the CUNone ratio, the higher performance that the PCK will bring.

```
gaussdb=> EXPLAIN PERFORMANCE
gaussdb-> SELECT * FROM orders_pck
gaussdb-> WHERE o_orderkey = '13095143'
gaussdb-> ORDER BY o_orderdate;

QUERY PLAN
-----
id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs
---+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
1 | --Row Adapter | 5.597 | 1 | 3 | 1 | 82KB | 1 | 123 | 94838.01
2 | --Vector Streaming (type: GATHER) | 5.589 | 1 | 3 | 1 | 825KB | 1 | 123 | 94838.01
3 | --Vector Sort | [1.858, 1.929] | 1 | 3 | 1 | [338KB, 411KB] | 16MB | [0,167] | 123 | 94838.01
4 | --CStore Scan on public.orders_pck | [1.742, 1.864] | 1 | 1 | 1 | [1MB, 1MB] | 1MB | 123 | 94838.00

Predicate Information (identified by plan id)
```

```
Datanode Information (identified by plan id)
-----
1 --Row Adapter
  (actual time=5.597..5.597 rows=1 loops=1)
  (CPU: ex c/r=815, ex row=1, ex cyc=559741)
2 --Vector Streaming (type: GATHER)
  (actual time=5.589..5.589 rows=1 loops=1)
  (Buffers: shared hit=3)
  (CPU: ex c/r=558926, ex row=1, ex cyc=558926, inc cyc=558926)
3 --Vector Sort
  dn_6001_6002 (actual time=1.858..1.858 rows=0 loops=1)
  dn_6003_6004 (actual time=1.914..1.914 rows=1 loops=1)
  dn_6005_6006 (actual time=1.929..1.929 rows=0 loops=1)
  dn_6001_6002 (Buffers: shared hit=305)
  dn_6003_6004 (Buffers: shared hit=306)
  dn_6005_6006 (Buffers: shared hit=306)
  dn_6001_6002 (CPU: ex c/r=0, ex row=0, ex cyc=11573, inc cyc=185784)
  dn_6003_6004 (CPU: ex c/r=12187, ex row=1, ex cyc=12187, inc cyc=191428)
  dn_6005_6006 (CPU: ex c/r=0, ex row=0, ex cyc=12455, inc cyc=192864)
4 --CStore Scan on public.orders_pck
  dn_6001_6002 (actual time=1.742..1.742 rows=0 loops=1) (filter time=0.497) (RoughCheck CU: CUNone: 82, CUSome: 2)
  dn_6003_6004 (actual time=1.694..1.793 rows=1 loops=1) (filter time=0.509) (RoughCheck CU: CUNone: 82, CUSome: 2)
  dn_6005_6006 (actual time=1.804..1.804 rows=0 loops=1) (filter time=0.509) (RoughCheck CU: CUNone: 82, CUSome: 2)
  dn_6001_6002 (Buffers: shared hit=392)
  dn_6003_6004 (Buffers: shared hit=393)
  dn_6005_6006 (Buffers: shared hit=393)
  dn_6001_6002 (CPU: ex c/r=0, ex row=5807635, ex cyc=174211, inc cyc=174211)
  dn_6003_6004 (CPU: ex c/r=0, ex row=5802975, ex cyc=179233, inc cyc=179233)
  dn_6005_6006 (CPU: ex c/r=0, ex row=4989390, ex cyc=180409, inc cyc=180409)
```

16.5.6 Case: Adjusting the Table Storage Mode in a Medium Table

In DWS, row-store tables use the row execution engine, and column-store tables use the column execution engine. If both row-store table and column-store tables exist in a SQL statement, the system will automatically select the row execution engine. The performance of a column execution engine (except for the indexscan related operators) is much better than that of a row execution engine. Therefore, a column-store table is recommended. This is important for some medium result set dumping tables, and you need to select a proper table storage type.

Before Optimization

During the test at a site, if the following execution plan is performed, the customer expects that the performance can be improved and the result can be returned within 3s.

| id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs |
|----|---|---------------------|----------|-----------|------------------|----------|---------|---------|-----------|
| 1 | Streaming (type: GATHER) | 4681.039 | 7 | 17 | 304KB | 1MB | | 41 | 101740.19 |
| 2 | Hash Join (3,7) | [4680.4429,889] | 7 | 17 | [88KB, 89KB] | 1MB | | 41 | 101739.10 |
| 3 | Append | [2474.514,3840.836] | 34782433 | 108109436 | [1KB, 1KB] | 1MB | | 49 | 88969.76 |
| 4 | Row Adapter | [2792.501,3040.924] | 33011417 | 99088596 | [49KB, 49KB] | 1MB | | 49 | 70615.19 |
| 5 | Partitioned CScan on sd_data.acct_account_his ta | [2288.421,2868.707] | 33011417 | 99088596 | [1002KB, 1012KB] | 1MB | | 49 | 70615.19 |
| 6 | Seq Scan on cscan_pg_delta_2428217623 ta | [163.977,169.707] | 1741016 | 9010840 | [13KB, 13KB] | 1MB | | 50 | 18354.66 |
| 7 | Hash | [4.385,7.999] | 9 | 32 | [260KB, 292KB] | 16MB | [0, 36] | 30 | 100.17 |
| 8 | Streaming (type: REDISTRIBUTE) | [4.384,7.997] | 9 | 32 | [1054KB, 1058KB] | 1MB | | 30 | 100.17 |
| 9 | Hash Join (10,11) | [0.162,1.043] | 9 | 32 | [8KB, 8KB] | 1MB | | 30 | 100.06 |
| 10 | Seq Scan on pg_temp_on_5001_140148717123928.input_acct_id_tbl tbl | [0.001,0.176] | 1030 | 31968 | [113KB, 113KB] | 1MB | | 11 | 15.99 |
| 11 | Hash | [0.001,0.849] | 9 | 32 | [268KB, 290KB] | 16MB | [0, 37] | 19 | 80.81 |
| 12 | HashAggregate | [0.001,0.849] | 9 | 32 | [10KB, 13KB] | 1MB | | 19 | 80.30 |
| 13 | Seq Scan on public.row_unlogged_table | [0.000,0.847] | 449 | 449 | [13KB, 13KB] | 1MB | | 19 | 78.70 |

(13 rows)

After Optimization

It is found that the row engine is used after analysis, because both the temporary plan table input_acct_id_tbl and the medium result dumping table row_unlogged_table use a row-store table.

After the two tables are changed into column-store tables, the system performance is improved and the result is returned by 1.6s.

| id | operation | A-time | A-rows | E-rows | Peak Memory | E-memory | A-width | E-width | E-costs |
|----|--|--------------------|---------|-----------|------------------|----------|---------|---------|-----------|
| 1 | Row Adapter | 1567.967 | 7 | 17 | 399KB | | | 41 | 101758.52 |
| 2 | Vector Streaming (type: GATHER) | 1567.949 | 7 | 17 | 399KB | | | 41 | 101758.52 |
| 3 | Vector Hash Join (4,8) | [8.130,1629.101] | 7 | 17 | [2362KB, 2446KB] | 16MB | | 41 | 101757.48 |
| 4 | Vector Append | [542.822,1462.479] | 6481770 | 108109436 | [1KB, 1KB] | 1MB | | 49 | 88969.76 |
| 5 | Partitioned CScan on sd_data.acct_account_his ta | [235.786,1155.830] | 3940754 | 99088596 | [861KB, 1012KB] | 1MB | | 49 | 70615.19 |
| 6 | Vector Adapter | [236.065,260.284] | 1741016 | 9010840 | [129KB, 129KB] | 1MB | | 50 | 18354.66 |
| 7 | Seq Scan on cscan_pg_delta_2428217623 ta | [182.895,168.048] | 1741016 | 9010840 | [13KB, 13KB] | 1MB | | 50 | 18354.66 |
| 8 | Vector Streaming (type: REDISTRIBUTE) | [7.727,12.981] | 9 | 32 | [1092KB, 1141KB] | 1MB | [0, 40] | 30 | 118.66 |
| 9 | Vector Hash Join (10,11) | [0.182,4.385] | 9 | 32 | [1217KB, 2217KB] | 16MB | | 30 | 118.45 |
| 10 | CStore Scan on pg_temp_on_5001_140148155066112.input_acct_id_tbl tbl | [4.372,4.372] | 999 | 31968 | [207KB, 207KB] | 1MB | | 11 | 81.00 |
| 11 | Vector Hash Aggregate | [0.062,0.209] | 9 | 32 | [223KB, 223KB] | 16MB | [0, 35] | 19 | 33.67 |
| 12 | CStore Scan on public.col_unlogged_table | [0.011,0.107] | 449 | 449 | [641KB, 596KB] | 1MB | | 19 | 32.08 |

16.5.7 Case: Reconstructing Partition Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Generally, partitioning is applied to tables that have obvious ranges. Partitions on such tables allow scanning on a small part of data, improving the query performance.

During query, partition pruning is used to minimize bottom-layer data scanning to narrow down the overall scope of scanning in a table. Partition pruning means that the optimizer can automatically extract partitions to be scanned based on the partition key specified in the **FROM** and **WHERE** statements. This avoids full table scanning, reduces the number of data blocks to be scanned, and improves performance.

Before Optimization

Create a non-partition table **orders_no_part**. The table definition is as follows:

```
pg_get_tabledef
-----
SET search_path = dbadmin;
CREATE TABLE orders_no_part (
  o_orderkey bigint NOT NULL,
  o_custkey bigint NOT NULL,
  o_orderstatus character(1) NOT NULL,
  o_totalprice numeric(15,2) NOT NULL,
  o_orderdate timestamp(0) without time zone NOT NULL,
  o_orderpriority character(15) NOT NULL,
  o_clerk character(15) NOT NULL,
  o_shippriority bigint NOT NULL,
  o_comment character varying(79) NOT NULL
)
WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)
DISTRIBUTE BY HASH(o_orderkey)
TO GROUP group_version1;
(1 row)
```

Run the following SQL statement to query the execution plan of the non-partition table:

```
EXPLAIN PERFORMANCE
SELECT count(*) FROM orders_no_part WHERE
o_orderdate >= '1996-01-01 00:00:00'::timestamp(0);
```

As shown in the following figure, the execution time is 73 milliseconds, and the full table scanning time is 44 to 45 milliseconds.

```
gaussdb> EXPLAIN PERFORMANCE
gaussdb-> SELECT count(*) FROM orders_no_part WHERE
gaussdb-> o_orderdate >= '1996-01-01 00:00:00'::timestamp(0);
```

| id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs |
|----|---|------------------|---------|---------|------------|----------------|----------|---------|---------|----------|
| 1 | -> Row Adapter | 73.623 | 1 | 1 | | 18KB | | | 8 | 99791.27 |
| 2 | -> Vector Aggregate | 73.611 | 1 | 1 | | 177KB | | | 8 | 99791.27 |
| 3 | -> Vector Streaming (type: GATHER) | 73.575 | 3 | 3 | | 80KB | | | 8 | 99791.27 |
| 4 | -> Vector Aggregate | [24.963, 55.561] | 3 | 3 | | [138KB, 138KB] | 1MB | | 8 | 99783.27 |
| 5 | -> CStore Scan on public.orders_no_part | [44.572, 45.877] | 5898663 | 5943988 | | [388KB, 388KB] | 1MB | | 8 | 94838.88 |

After Optimization

Create a partitioned table **orders**. The table is defined as follows:

```
pg_get_tabledef
-----
SET search_path = dbadmin;
CREATE TABLE orders (
  o_orderkey bigint NOT NULL,
  o_custkey bigint NOT NULL,
  o_orderstatus character(1) NOT NULL,
  o_totalprice numeric(15,2) NOT NULL,
  o_orderdate timestamp(0) without time zone NOT NULL,
  o_orderpriority character(15) NOT NULL,
  o_clerk character(15) NOT NULL,
  o_shippriority bigint NOT NULL,
  o_comment character varying(79) NOT NULL
)
WITH (orientation=column, compression=low, colversion=2.0, enable_delta=false)
DISTRIBUTE BY HASH(o_orderkey)
TO GROUP group_version1
PARTITION BY RANGE (o_orderdate)
(
  PARTITION o_orderdate_1 VALUES LESS THAN ('1993-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_2 VALUES LESS THAN ('1994-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_3 VALUES LESS THAN ('1995-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_4 VALUES LESS THAN ('1996-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_5 VALUES LESS THAN ('1997-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_6 VALUES LESS THAN ('1998-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default,
  PARTITION o_orderdate_7 VALUES LESS THAN ('1999-01-01 00:00:00'::timestamp(0) without time zone) TABLESPACE pg_default
)
ENABLE ROW MOVEMENT;
(1 row)
```

Run the SQL statement again to query the execution plan of the partitioned table. The execution time is 40 ms, in which the table scanning time is only 13 ms. The smaller the value of **Iterations**, the better the partition pruning effect.

```
EXPLAIN PERFORMANCE
SELECT count(*) FROM orders_no_part WHERE
o_orderdate >= '1996-01-01 00:00:00'::timestamp(0);
```

As shown in the following figure, the execution time is 40 milliseconds, and the table scanning time is only 13 milliseconds. A smaller **Iterations** value indicates a better partition pruning effect.

| id | operation | A-time | A-rows | E-rows | E-distinct | Peak Memory | E-memory | A-width | E-width | E-costs |
|----|---|-------------------|--------|---------|------------|----------------|----------|---------|---------|----------|
| 1 | -> Row Adapter | 40.925 | 1 | 1 | 1 | 19KB | | | 8 | 22382.64 |
| 2 | -> Vector Aggregate | 40.915 | 1 | 1 | 1 | 177KB | | | 8 | 22382.64 |
| 3 | -> Vector Streaming (type: GATHER) | 40.915 | 3 | 3 | 3 | 89KB | | | 8 | 22382.64 |
| 4 | -> Vector Aggregate | [20.987, 21.229] | 3 | 3 | 3 | [138KB, 138KB] | 1MB | | 8 | 22374.64 |
| 5 | -> Vector Partition Iterator | [13.734, 13.939] | 588865 | 5848363 | | [17KB, 17KB] | 1MB | | 8 | 17581.88 |
| 6 | -> Partitioned CStore Scan on public.orders | [13.895, 13.3793] | 588865 | 5848363 | | [299KB, 299KB] | 1MB | | 8 | 17591.88 |

Predicate Information (identified by plan id)

5 --Vector Partition Iterator
Iterations: 3

6 --Partitioned CStore Scan on public.orders
Filter: (orders.o_orderdate >= '1996-01-01 00:00:00'::timestamp(0) without time zone)
Partitions Selected by Static Prune: 6..7

16.5.8 Case: Adjusting the GUC Parameter best_agg_plan

Symptom

The t1 table is defined as follows:

```
create table t1(a int, b int, c int) distribute by hash(a);
```

Assume that the distribution column of the result set provided by the agg lower-layer operator is setA, and the group by column of the agg operation is setB, the agg operations can be performed in two scenarios in the stream framework.

Scenario 1: setA is a subset of setB.

In this scenario, the aggregation result of the lower-layer result set is the correct result, which can be directly used by the upper-layer operator. For details, see the following figure:

```
explain select a, count(1) from t1 group by a;
```

| id | operation | E-rows | E-width | E-costs |
|----|-----------------------------|--------|---------|---------|
| 1 | -> Streaming (type: GATHER) | 30 | 4 | 15.56 |
| 2 | -> HashAggregate | 30 | 4 | 14.31 |
| 3 | -> Seq Scan on t1 | 30 | 4 | 14.14 |

(3 rows)

Scenario 2: setA is not a subset of setB.

In this scenario, the Stream execution framework is classified into the following three plans:

hashagg+gather(redistribute)+hashagg

redistribute+hashagg(+gather)

hashagg+redistribute+hashagg(+gather)

DWS provides the guc parameter **best_agg_plan** to intervene the execution plan, and forces the plan to generate the corresponding execution plan. This parameter can be set to 0, 1, 2, and 3.

- When the value is set to 1, the first plan is forcibly generated.
- When the value is set to 2 and if the **group by** column can be redistributed, the second plan is forcibly generated. Otherwise, the first plan is generated.

- When the value is set to **3** and if the **group by** column can be redistributed, the third plan is generated. Otherwise, the first plan is generated.
- When the value is set to **0**, the query optimizer chooses the most optimal plan by the three preceding plans' evaluation cost.

Possible impacts are as follows:

```
set best_agg_plan to 1;
SET
explain select b,count(1) from t1 group by b;
id |          operation          | E-rows | E-width | E-costs
+-----+-----+-----+-----+
1 | -> HashAggregate           |      8 |      4 | 15.83
2 | -> Streaming (type: GATHER) |     25 |      4 | 15.83
3 | -> HashAggregate           |     25 |      4 | 14.33
4 | -> Seq Scan on t1          |     30 |      4 | 14.14
(4 rows)
set best_agg_plan to 2;
SET
explain select b,count(1) from t1 group by b;
id |          operation          | E-rows | E-width | E-costs
+-----+-----+-----+-----+
1 | -> Streaming (type: GATHER) |     30 |      4 | 15.85
2 | -> HashAggregate           |     30 |      4 | 14.60
3 | -> Streaming(type: REDISTRIBUTE) |     30 |      4 | 14.45
4 | -> Seq Scan on t1          |     30 |      4 | 14.14
(4 rows)
set best_agg_plan to 3;
SET
explain select b,count(1) from t1 group by b;
id |          operation          | E-rows | E-width | E-costs
+-----+-----+-----+-----+
1 | -> Streaming (type: GATHER) |     30 |      4 | 15.84
2 | -> HashAggregate           |     30 |      4 | 14.59
3 | -> Streaming(type: REDISTRIBUTE) |     25 |      4 | 14.59
4 | -> HashAggregate           |     25 |      4 | 14.33
5 | -> Seq Scan on t1          |     30 |      4 | 14.14
(5 rows)
```

Summary

Generally, the optimizer chooses an optimal execution plan, but the cost estimation, especially that of the intermediate result set, has large deviations, which may result in large deviations in agg calculation. In this case, you need to use `best_agg_plan` to adjust the agg calculation model.

When the aggregation convergence ratio is very small, that is, the number of result sets does not become small obviously after the agg operation (5 times is a critical point), you can select the redistribute+hashagg or hashagg+redistribute+hashagg execution mode.

16.5.9 Case: Rewriting SQL Statements and Eliminating Prune Interference

A filter criterion that contains the expression of partition key cannot be used for pruning. As a result, the query statement scans almost all data in the partitioned table.

Before Optimization

t_ddw_f10_op_cust_asset_mon indicates the partitioned table. **year_mth** indicates the partition key. This field is an integer consisting of the **year** and **mth** values.

The following figure shows the tested SQL statements.

```
SELECT
  count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1,6)
AND b1.year_mth + 1 >= substr('20200722',1,6);
```

The test result shows that the table scan of the SQL statement takes 10 seconds. The execution plan of the SQL statement is as follows.

```
EXPLAIN (ANALYZE ON, VERBOSE ON)
SELECT
  count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1,6)
AND b1.year_mth + 1 >= cast(substr('20200722',1,6) AS int);
```

QUERY PLAN

| id | operation | A-time | A-rows | E-rows | E- |
|----------|--|-----------------------|--------------------|-----------|-----------|
| distinct | Peak Memory | E-memory | A-width | E-width | E-costs |
| 1 | -> Aggregate | 10662.260 | 1 | 1 | |
| 32KB | | | 8 | 593656.42 | |
| 2 | -> Streaming (type: GATHER) | 10662.172 | | 4 | 4 |
| | 136KB | | 8 | 593656.42 | |
| 3 | -> Aggregate | [9692.785, 10656.068] | 4 | 4 | |
| | [24KB, 24KB] 1MB | | 8 | 593646.42 | |
| 4 | -> Partition Iterator | [8787.198, 9629.138] | 16384000 | | |
| 32752850 | | [16KB, 16KB] 1MB | | 0 | 573175.88 |
| 5 | -> Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1 | [8365.655, 9152.115] | | | |
| 16384000 | 32752850 | | [32KB, 32KB] 1MB | | 0 |
| | | | | | 573175.88 |

SQL Diagnostic Information

Partitioned table unprunable Qual
table public.t_ddw_f10_op_cust_asset_mon b1:
left side of expression "((year_mth + 1) > 202008)" invokes function-call/type-conversion

Predicate Information (identified by plan id)

4 --Partition Iterator
Iterations: 6

5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
Filter: ((b1.year_mth < 202007::bigint) AND ((b1.year_mth + 1) >= 202007))
Rows Removed by Filter: 81920000
Partitions Selected by Static Prune: 1..6

After Optimization

After analyzing the execution plan of the statement and checking the SQL self-diagnosis information in the execution plan, the following diagnosis information is found:

SQL Diagnostic Information

Partitioned table unprunable Qual
table public.t_ddw_f10_op_cust_asset_mon b1:
left side of expression "((year_mth + 1) > 202008)" invokes function-call/type-conversion

The filter criterion contains the expression **(year_mth + 1) > 202008**. A filter criterion that contains the expression of partition key cannot be used for pruning. As a result, the query statement scans almost all data in the partitioned table.

Compared with the original SQL statement, the expression **(year_mth + 1) > 202008** is derived from the expression **b1.year_mth + 1 > substr('20200822',1,6)**. Based on the diagnosis information, the SQL statement is modified as follows.

```
SELECT
  count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth <= substr('20200822',1,6)
AND b1.year_mth > cast(substr('20200822',1,6) AS int) - 1;
```

After the modification, the SQL statement execution information is as follows. The alarm indicating that the pruning is not performed is cleared. After the pruning, the score of the partition to be scanned is 1, and the execution time is shortened from 10 seconds to 3 seconds.

```
EXPLAIN (analyze ON, verbose ON)
SELECT
  count(1)
FROM t_ddw_f10_op_cust_asset_mon b1
WHERE b1.year_mth < substr('20200722',1,6)
AND b1.year_mth >= cast(substr('20200722',1,6) AS int) - 1;
```

QUERY PLAN

| id | operation | A-time | A-rows | E-rows | E- |
|----------|--|----------------------|----------|---------|---------|
| distinct | Peak Memory | E-memory | A-width | E-width | E-costs |
| 1 | -> Aggregate | 3009.796 | 1 | 1 | |
| 2 | -> Streaming (type: GATHER) | 3009.718 | | 4 | 4 |
| 3 | -> Aggregate | [2675.509, 3003.298] | | 4 | 4 |
| 4 | -> Partition Iterator | [1820.725, 2053.836] | 16384000 | | |
| 5 | -> Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1 | [1420.972, 1590.083] | | | |

Predicate Information (identified by plan id)

```
4 --Partition Iterator
  Iterations: 1
5 --Partitioned Seq Scan on public.t_ddw_f10_op_cust_asset_mon b1
  Filter: ((b1.year_mth < 202007::bigint) AND (b1.year_mth >= 202006))
  Partitions Selected by Static Prune: 6
```

16.5.10 Case: Rewriting SQL Statements and Deleting in-clause

Before Optimization

in-clause/any-clause is a common SQL statement constraint. Sometimes, the clause following **in** or **any** is a constant. For example:

```
select
  count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in ('20120405', '20130405');
```

or

```
select
count(1)
from calc_empfyc_c1_result_tmp_t1
where ls_pid_cusr1 in any('20120405', '20130405');
```

Some special usages are as follows:

```
SELECT
ls_pid_cusr1, COALESCE(max(round((current_date-bthdate)/365)),0)
FROM calc_empfyc_c1_result_tmp_t1 t1, p10_md_tmp_t2 t2
WHERE t1.ls_pid_cusr1 = any(values(id), (id15))
GROUP BY ls_pid_cusr1;
```

Where **id** and **id15** are columns of p10_md_tmp_t2. ls_pid_cusr1 = any(values(id), (id15)) equals t1. ls_pid_cusr1 = id or t1. ls_pid_cusr1 = id15.

Therefore, join-condition is essentially an inequality, and nestloop must be used for this join operation. The execution plan is as follows:

```
Streaming (type: GATHER) (cost=1641429284.14..1641429523.98 rows=3840 width=49)
Node/s: All datanodes
-> Insert on channel calc_empfyc_c1_result_age_tmp (cost=1641429280.14..1641429283.98 rows=3840 width=49)
-> HashAggregate (cost=1641429280.14..1641429283.98 rows=3840 width=25)
Output: t1.ls_pid_cusr1, COALESCE(max(max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365)::double precision))::numeric, 0))), 0)::numeric)
Group By Key: t1.ls_pid_cusr1
-> Streaming (type: REDISTRIBUTE) (cost=820714640.07..820714642.69 rows=3968 width=25)
Output: t1.ls_pid_cusr1, max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365)::double precision))::numeric, 0))
Distribute Key: t1.ls_pid_cusr1
Spawn on: All datanodes
-> HashAggregate (cost=820714640.07..820714642.69 rows=3968 width=25)
Output: t1.ls_pid_cusr1, max(round(((('2017-03-29 00:00:00'::timestamp without time zone - t2.bthdate) / 365)::double precision))::numeric, 0))
Group By Key: t1.ls_pid_cusr1
-> Nested Loop (cost=0.00..415567760.93 rows=875293350960 width=25)
Output: t1.ls_pid_cusr1, t2.bthdate
Join Filter: (SubPlan 1)
-> Seq Scan on channel.p10_md_tmp_t2 t2 (cost=0.00..127030.52 rows=443523360 width=64)
Output: t2.id, t2.id15, t2.bthdate, t2.randeq
-> Materialize (cost=0.00..147.29 rows=252608 width=17)
Output: t1.ls_pid_cusr1
-> Streaming (type: BROADCAST) (cost=0.00..127.56 rows=252608 width=17)
Output: t1.ls_pid_cusr1
Spawn on: All datanodes
-> Seq Scan on channel.calc_empfyc_c1_result_tmp_t1 t1 (cost=0.00..1.62 rows=3947 width=17)
Output: t1.ls_pid_cusr1
SubPlan 1
-> Values Scan on "VALUES" (cost=0.00..0.01 rows=64 width=38)
Output: "VALUES".column1
```

After Optimization

The test result shows that both result sets are too large. As a result, nestloop is time-consuming with more than one hour to return results. Therefore, the key to performance optimization is to eliminate nestloop, using more efficient hashjoin. From the perspective of semantic equivalence, the SQL statements can be written as follows:

```
select
ls_pid_cusr1, COALESCE(max(round(ym/365)),0)
from
(
    (
        SELECT
        ls_pid_cusr1, (current_date-bthdate) as ym
        FROM calc_empfyc_c1_result_tmp_t1 t1, p10_md_tmp_t2 t2
        WHERE t1.ls_pid_cusr1 = t2.id and t1.ls_pid_cusr1 != t2.id15
    )
    union all
    (
        SELECT
        ls_pid_cusr1, (current_date-bthdate) as ym
        FROM calc_empfyc_c1_result_tmp_t1 t1, p10_md_tmp_t2 t2
        WHERE t1.ls_pid_cusr1 = id15
    )
)
GROUP BY ls_pid_cusr1;
```

Note: Use **UNION ALL** instead of **UNION** if possible. **UNION** eliminates duplicate rows while merging two result sets but **UNION ALL** merges the two result sets

without deduplication. Therefore, replace **UNION** with **UNION ALL** if you are sure that the two result sets do not contain duplicate rows based on the service logic.

The optimized SQL queries consist of two equivalent join subqueries, and each subquery can be used for hashjoin in this scenario. The optimized execution plan is as follows:

| id | operation | A-time | A-rows | B-rows | Peak Memory | E-memory | A-width |
|-----------|--|---------------------|-----------|-----------|------------------|----------|---------|
| 1 | -> Streaming (type: GATHER) | 6737.281 | 0 | 192 | 292KB | | |
| 2 | -> Insert on channel.calc_empfyc_cl_result_age_tmp | [4665.024,4990.666] | 0 | 192 | [1108KB, 1108KB] | 1MB | |
| 3 | -> HashAggregate | [4664.996,4990.641] | 0 | 192 | [12KB, 12KB] | 16MB | |
| 4 | -> Streaming (type: REDISTRIBUTE) | [4664.991,4990.637] | 0 | 3392 | [2090KB, 2090KB] | 1MB | |
| 5 | -> HashAggregate | [3816.939,4958.348] | 0 | 3392 | [14KB, 14KB] | 16MB | |
| 6 | -> Append | [3816.936,4958.340] | 0 | 4011 | [1KB, 1KB] | 1MB | |
| 7 | -> Hash Join (8,9) | [2011.226,3080.697] | 0 | 3947 | [6KB, 6KB] | 1MB | |
| 8 | -> Seq Scan on channel.pi0_md_tmp_t2 t2 | [803.782,1238.984] | 443525717 | 443523360 | [12KB, 12KB] | 1MB | |
| 9 | -> Hash | [4.357,328.979] | 252608 | 252608 | [482KB, 482KB] | 16MB | [35,39] |
| 10 | -> Streaming (type: BROADCAST) | [2.345,326.320] | 252608 | 252608 | [2090KB, 2090KB] | 1MB | |
| 11 | -> Seq Scan on channel.calc_empfyc_cl_result_tmp_t1 t1 | [0.011,0.030] | 3947 | 3947 | [11KB, 11KB] | 1MB | |
| 12 | -> Hash Join (13,14) | [1376.258,2066.110] | 0 | 64 | [5KB, 5KB] | 1MB | |
| 13 | -> Seq Scan on channel.pi0_md_tmp_t2 t2 | [777.552,1388.499] | 443525717 | 443523360 | [12KB, 12KB] | 1MB | |
| 14 | -> Hash | [2.812,4.217] | 252608 | 252608 | [482KB, 482KB] | 16MB | [25,27] |
| 15 | -> Streaming (type: BROADCAST) | [1.276,1.868] | 252608 | 252608 | [2090KB, 2090KB] | 1MB | |
| 16 | -> Seq Scan on channel.calc_empfyc_cl_result_tmp_t1 t1 | [0.010,0.033] | 3947 | 3947 | [11KB, 11KB] | 1MB | |
| (16 rows) | | | | | | | |

Before the optimization, no result is returned for more than 1 hour. After the optimization, the result is returned within 7s.

16.5.11 Case: Setting Partial Cluster Keys

You can add **PARTIAL CLUSTER KEY**(*column_name*[,...]) to the definition of a column-store table to set one or more columns of this table as partial cluster keys. In this way, each 70 CUs (4.2 million rows) will be sorted based on the cluster keys by default during data import and the value range is narrowed down for each of the new 70 CUs. If the **where** condition in the query statement contains these columns, the filtering performance will be improved.

Before Optimization

The partial cluster key is not used. The table is defined as follows:

```
CREATE TABLE lineitem
(
  L_ORDERKEY BIGINT NOT NULL
, L_PARTKEY BIGINT NOT NULL
, L_SUPPKEY BIGINT NOT NULL
, L_LINENUMBER BIGINT NOT NULL
, L_QUANTITY DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL
, L_DISCOUNT DECIMAL(15,2) NOT NULL
, L_TAX DECIMAL(15,2) NOT NULL
, L_RETURNFLAG CHAR(1) NOT NULL
, L_LINESTATUS CHAR(1) NOT NULL
, L_SHIPDATE DATE NOT NULL
, L_COMMITDATE DATE NOT NULL
, L_RECEIPTDATE DATE NOT NULL
, L_SHIPINSTRUCT CHAR(25) NOT NULL
, L_SHIPMODE CHAR(10) NOT NULL
, L_COMMENT VARCHAR(44) NOT NULL
)
with (orientation = column)
distribute by hash(L_ORDERKEY);

select
sum(L_extendedprice * L_discount) as revenue
from
lineitem
where
L_shipdate >= '1994-01-01'::date
and L_shipdate < '1994-01-01'::date + interval '1 year'
and L_discount between 0.06 - 0.01 and 0.06 + 0.01
and L_quantity < 24;
```

After the data is imported, perform the query and check the execution time.

Figure 16-23 Partial cluster keys not used

| id | operation | A-time | A-rows | E-rows | Peak Memory | A-width | E-width | E-costs |
|----|------------------------------------|----------------------|--------|--------|----------------|---------|---------|-----------|
| 1 | -> Row Adapter | 1653.156 | 1 | 1 | 12KB | | 44 | 205803.90 |
| 2 | -> Vector Aggregate | 1653.146 | 1 | 1 | 184KB | | 44 | 205803.90 |
| 3 | -> Vector Streaming (type: GATHER) | 1653.070 | 1 | 1 | 174KB | | 44 | 205803.90 |
| 4 | -> Vector Aggregate | [1481.497, 1481.497] | 1 | 1 | [225KB, 225KB] | | 44 | 205803.84 |
| 5 | -> CStore Scan on public.lineitem | [1405.004, 1405.004] | 114160 | 111485 | [792KB, 792KB] | | 12 | 205246.40 |

Figure 16-24 CU loading without partial cluster keys

```

5 --CStore Scan on public.lineitem
  datanode1 (actual time=40.623..1405.004 rows=114160 loops=1)
    datanode1 (RoughCheck CU: CUNone: 0, CUSome: 101)
    datanode1 (LLVM Optimized)
    datanode1 (Buffers: shared hit=18385 read=23)
    datanode1 (CPU: ex c/r=31917, ex cyc=3643646206, inc cyc=3643646206)

```

After Optimization

In the **where** condition, both the **L_shipdate** and **L_quantity** columns have a few distinct values, and their values can be used for min/max filtering. Therefore, modify the table definition as follows:

```

CREATE TABLE lineitem
(
  L_ORDERKEY BIGINT NOT NULL
, L_PARTKEY BIGINT NOT NULL
, L_SUPPKEY BIGINT NOT NULL
, L_LINENUMBER BIGINT NOT NULL
, L_QUANTITY DECIMAL(15,2) NOT NULL
, L_EXTENDEDPRICE DECIMAL(15,2) NOT NULL
, L_DISCOUNT DECIMAL(15,2) NOT NULL
, L_TAX DECIMAL(15,2) NOT NULL
, L_RETURNFLAG CHAR(1) NOT NULL
, L_LINESTATUS CHAR(1) NOT NULL
, L_SHIPDATE DATE NOT NULL
, L_COMMITDATE DATE NOT NULL
, L_RECEIPTDATE DATE NOT NULL
, L_SHIPINSTRUCT CHAR(25) NOT NULL
, L_SHIPMODE CHAR(10) NOT NULL
, L_COMMENT VARCHAR(44) NOT NULL
, partial cluster key(L_shipdate, L_quantity)
)
with (orientation = column)
distribute by hash(L_ORDERKEY);

```

Import the data again, perform the query, and check the execution time.

Figure 16-25 Partial cluster keys used

| id | operation | A-time | A-rows | E-rows | Peak Memory | A-width | E-width | E-costs |
|----|------------------------------------|--------------------|--------|--------|----------------|---------|---------|-----------|
| 1 | -> Row Adapter | 459.539 | 1 | 1 | 12KB | | 44 | 205693.85 |
| 2 | -> Vector Aggregate | 459.528 | 1 | 1 | 184KB | | 44 | 205693.85 |
| 3 | -> Vector Streaming (type: GATHER) | 459.452 | 1 | 1 | 174KB | | 44 | 205693.85 |
| 4 | -> Vector Aggregate | [285.177, 285.177] | 1 | 1 | [225KB, 225KB] | | 44 | 205693.79 |
| 5 | -> CStore Scan on public.lineitem | [249.757, 249.757] | 114160 | 89475 | [792KB, 792KB] | | 12 | 205246.40 |

Figure 16-26 CU loading with partial cluster keys

```

5 --CStore Scan on public.lineitem
  datanode1 (actual time=23.017..249.757 rows=114160 loops=1)
    datanode1 (RoughCheck CU: CUNone: 84, CUSome: 17)
    datanode1 (LLVM Optimized)
    datanode1 (Buffers: shared hit=2853 read=23)
    datanode1 (CPU: ex c/r=5673, ex cyc=647656146, inc cyc=647656146)

```

After partial cluster keys are used, the execution time of **5-- CStore Scan on public.lineitem** decreases by 1.2s because 84 CUs are filtered out.

Optimization

- Select partial cluster keys.
 - The following data types support cluster keys: character varying(n), varchar(n), character(n), char(n), text, nvarchar2, timestamp with time zone, timestamp without time zone, date, time without time zone, and time with time zone.
 - Smaller number of distinct values in a partial cluster key generates higher filtering performance.
 - Columns that can filter out larger amount of data is preferentially selected as partial cluster keys.
 - If multiple columns are selected as partial cluster keys, the columns are used in sequence to sort data. You are advised to select a maximum of three columns.

- Modify parameters to reduce the impact of partial cluster keys on the import performance.

After partial cluster keys are used, data will be sorted when they are imported, affecting the import performance. If all the data can be sorted in the memory, the keys have little impact on import. If some data cannot be sorted in the memory and is written into a temporary file for sorting, the import performance will be greatly affected.

The memory used for sorting is specified by the **psort_work_mem** parameter. You can set it to a larger value so that the sorting has less impact on the import performance.

The volume of data to be sorted is specified by the **PARTIAL_CLUSTER_ROWS** parameter of the table. Decreasing the value of this parameter reduces the amount of data to be sorted at a time. **PARTIAL_CLUSTER_ROWS** is usually used along with the **MAX_BATCHROW** parameter. The value of **PARTIAL_CLUSTER_ROWS** must be an integer multiple of the **MAX_BATCHROW** value. **MAX_BATCHROW** specifies the maximum number of rows in a CU.

16.5.12 Case: Converting from NOT IN to NOT EXISTS

nestloop anti join must be used to implement **NOT IN**, while you can use **Hash anti join** to implement **NOT EXISTS**. If no **NULL** value exists in the **JOIN** column, **NOT IN** is equivalent to **NOT EXISTS**. Therefore, if you are sure that no **NULL** value exists, you can convert **NOT IN** to **NOT EXISTS** to generate **hash joins** and to improve the query performance.

Before Optimization

Create two base tables **t1** and **t2**.

```
CREATE TABLE t1(a int, b int, c int not null) WITH(orientation=row);  
CREATE TABLE t2(a int, b int, c int not null) WITH(orientation=row);
```

Run the following SQL statement to query the **NOT IN** execution plan:

```
EXPLAIN VERBOSE SELECT * FROM t1 WHERE t1.c NOT IN (SELECT t2.c FROM t2);
```

The following figure shows the statement output.

| QUERY PLAN | | | | | |
|------------|---------------------------------|--------|------------|---------|---------|
| id | operation | E-rows | E-distinct | E-width | E-costs |
| 1 | -> Streaming (type: GATHER) | 6 | | 12 | 78.98 |
| 2 | -> Nested Loop Anti Join (3, 4) | 6 | | 12 | 64.98 |
| 3 | -> Seq Scan on public.t1 | 60 | | 12 | 18.18 |
| 4 | -> Materialize | 360 | | 4 | 30.75 |
| 5 | -> Streaming(type: BROADCAST) | 360 | | 4 | 30.45 |
| 6 | -> Seq Scan on public.t2 | 60 | | 4 | 18.18 |

Predicate Information (identified by plan id)

| | |
|---|--|
| 2 | --Nested Loop Anti Join (3, 4) |
| | Join Filter: ((t1.c = t2.c) OR (t1.c IS NULL) OR (t2.c IS NULL)) |

According to the returned result, nest loops are used. As the OR operation result of NULL and any value is NULL,

```
t1.c NOT IN (SELECT t2.c FROM t2)
```

the preceding condition expression is equivalent to:

```
t1.c <> ANY(t2.c) AND t1.c IS NOT NULL AND ANY(t2.c) IS NOT NULL
```

After Optimization

The query can be modified as follows:

```
SELECT * FROM t1 WHERE NOT EXISTS (SELECT * FROM t2 WHERE t2.c = t1.c);
```

Run the following statement to query the execution plan of **NOT EXISTS**:

```
EXPLAIN VERBOSE SELECT * FROM t1 WHERE NOT EXISTS (SELECT 1 FROM t2 WHERE t2.c = t1.c);
```

| QUERY PLAN | | | | | |
|------------|----------------------------------|--------|------------|---------|---------|
| id | operation | E-rows | E-distinct | E-width | E-costs |
| 1 | -> Streaming (type: GATHER) | 6 | | 12 | 54.56 |
| 2 | -> Hash Anti Join (3, 5) | 6 | | 12 | 40.56 |
| 3 | -> Streaming(type: REDISTRIBUTE) | 60 | 10 | 12 | 20.12 |
| 4 | -> Seq Scan on public.t1 | 60 | | 12 | 18.18 |
| 5 | -> Hash | 59 | 10 | 4 | 20.12 |
| 6 | -> Streaming(type: REDISTRIBUTE) | 60 | | 4 | 20.12 |
| 7 | -> Seq Scan on public.t2 | 60 | | 4 | 18.18 |

Predicate Information (identified by plan id)

| | |
|---|--------------------------|
| 2 | --Hash Anti Join (3, 5) |
| | Hash Cond: (t1.c = t2.c) |

17 Hybrid Data Warehouse

17.1 Introduction to Hybrid Data Warehouse

A hybrid data warehouse needs to work with data sources, such as upstream databases or applications, to insert, upsert, and update data in real time. The data warehouse should also be able to query data shortly after it was imported.

Currently, the existing row-store and column-store tables in a conventional DWS data warehouse cannot meet real-time data import and query requirements. Row-store tables have strong real-time import capabilities and support highly concurrent updates, but their disk usage is high and query efficiency is low. Column-store tables have high data compression ratio and good OLAP query performance, but do not support concurrent updates. Concurrent import will cause severe lock conflicts.

To solve these problems, we use column storage to reduce the disk usage, support highly concurrency updates, and improve query speed. DWS hybrid data warehouses use HStore tables to achieve high performance during real-time data import and query, and have the transaction processing capabilities required for traditional OLTP scenarios.

The HStore tables uniquely support single and small-batch real-time IUD operations, as well as regular large-batch import. Data can be queried immediately after being imported. You can deduplicate traditional indexes (such as primary keys) and accelerate point queries. You can further accelerate OLAP queries through partitioning, multi-dimensional dictionaries, and partial sorting. Strong data consistency can be ensured for transactions with heavy workloads, such as TPC-C.

 NOTE

- Only clusters 8.2.0.100 and later support the HStore tables of the hybrid data warehouse.
- HStore Opt tables are recommended for 9.1.0 and later versions. HStore tables can be replaced by **HStore_opt** tables for better performance, except in scenarios requiring high performance without micro-batch updates.
- The hybrid data warehouse is used for both production and analysis. It is suitable for hybrid transaction and analysis workloads. For details about how to create a hybrid data warehouse, see "Creating a DWS Storage-Compute Coupled 2.0 Cluster" in the *Data Warehouse Service (DWS) User Guide*.
- Hot and cold data management is supported for HStore tables. For details, see [DWS Hot and Cold Data Management](#). This function is supported only by cluster versions 8.2.0.101 and later.
- HStore is a table type designed for the hybrid data warehouse and is irrelevant to the SQL parameter **hstore**.

Differences from Standard Data Warehouses

Hybrid data warehouses and standard data warehouses are two types of DWS data warehouses with different specifications and usage. For details, see [Table 17-1](#).

Table 17-1 Comparison between hybrid and standard data warehouses

| Type | Standard Data Warehouse | Hybrid Data Warehouse |
|----------------------|---|---|
| Application scenario | Converged data analysis using OLAP. It is used in sectors such as finance, government and enterprise, e-commerce, and energy. | Real-time data import + Hybrid analysis. Real-time upstream data import + Real-time query after data import. It is mainly used in scenarios that have high requirements on real-time data import, such as e-commerce and finance. |
| Advantage | It is cost-effective and widely used.

Cost effective, both hot and cold data analysis supported, elastic storage and compute capacities. | Hybrid load, high data import performance.

It achieves high query efficiency and high data compression ratio that are equivalent to those of column storage. It can also process transactions in traditional OLTP scenarios. |

| Type | Standard Data Warehouse | Hybrid Data Warehouse |
|---------------|---|--|
| Features | Excellent performance in interactive analysis and offline processing of massive data, as well as complex data mining. | It supports highly concurrent update operations on massive amounts of data and can achieve high query efficiency. It achieves high performance when processing a large amount of data in scenarios like high-concurrency import and latency-sensitive queries. |
| SQL syntax | Highly compatible with SQL syntax | Compatible with column-store syntax |
| GUC parameter | You can configure a wide variety of GUC parameters to tailor your data warehouse environment. | It is compatible with standard data warehouse GUC parameters and supports hybrid data warehouse tuning parameters. |

Technical Highlights

- Transaction consistency
Data can be retrieved for queries immediately after being inserted or updated. After concurrent updates, data is strongly consistent, and there will not be incorrect results caused by wrong update sequence.
- High query performance
In complex OLAP queries, such as multi-table correlation, the data warehouse achieves high performance through comprehensive distributed query plans and distributed executors. It also supports complex subqueries and stored procedures.
- Quick import
There will not be lock conflicts on column-store CUs. High-concurrency update and import operations are supported. The concurrent update performance can be over 100 times higher than before in general scenarios.
- High compression
Column storage can achieve a high compression ratio. Data is stored in the column-store primary table through MERGE can be compressed to greatly reduce disk usage and I/O.
- Query acceleration

You can deduplicate traditional indexes (such as primary keys) and accelerate point queries. You can further accelerate OLAP queries through partitioning, multi-dimensional dictionaries, and partial sorting.

Comparison Between Row-store, Column-store, and HStore Tables

Table 17-2 Comparison between row-store, column-store, and HStore tables

| Table Type | Row-Store | Column-Store | HStore |
|-------------------|--|--|--|
| Data storage mode | The attributes of a tuple are stored nearby. | The values of an attribute are stored nearby in the unit of CU. | Data is stored in the column-store primary tables as CUs. Updated columns and data inserted in small batches is serialized and then stored in a newly designed delta table. |
| Data write | Row-store compression has not been put into commercial use. Data is stored as it is, occupying a large amount of disk space. | In row storage, data with the same attribute value types is easy to compress. Data write consumes much fewer I/O resources and less disk space. | Data inserted in batches is directly written to CUs, which are as easy to compress as column storage. Updated columns and data inserted in small batches are serialized and then compressed. They will also be periodically merged to primary table CUs. |
| Data update | Data is updated by row, avoiding CU lock conflicts. The performance of concurrent updates (UPDATE/UPSERT/DELETE) is high. | The entire CU needs to be locked even if only one record in it is updated. Generally, concurrent updates (UPDATE/UPSERT/DELETE) are not supported. | CU lock conflicts can be avoided. The performance of concurrent updates (UPDATE/UPSERT/DELETE) is higher than 60% of the row-store update performance. |

| Table Type | Row-Store | Column-Store | HStore |
|----------------------|---|---|--|
| Data read | Data is read by row. An entire row needs to be retrieved even if only one column in it needs to be accessed. The query performance is low. | When data is read by column, only the CU of a column needs to be accessed. CUs can be easily compressed, occupying less I/O resources, and achieve high read performance. | Data in a column-store primary table is read by column. Updated columns and data inserted in small batches are deserialized and then retrieved. After data is merged to the primary table, the data can be read as easily as that in column storage. |
| Advantage | The concurrent update performance is high. | The query performance is high, and the disk space usage is small. | The concurrent update performance is high. After data merges, the query and compression performance are the same as those of column storage. |
| Disadvantage | A large amount of disk space is occupied, and the query performance is low. | Generally, concurrent updates are not supported. | A background permanent thread is required to clear unnecessary HStore table data after merge. Data is merged to the primary table CUs and then cleared. This operation is irrelevant to the SQL syntax MERGE . |
| Application scenario | <ol style="list-style-type: none">OLTP transactions with frequent update and deletion operationsPoint queries (simple queries that are based on indexes and return a small amount of data) | <ol style="list-style-type: none">OLAP query and analysisA large volume of data is imported, and is rarely updated or deleted after the import. | <ol style="list-style-type: none">Data is concurrently imported to the database in real time.High-concurrency update and import; and high-performance query |

17.2 Support and Constraints

A hybrid data warehouse is compatible with all column-store syntax.

Table 17-3 Supported syntax

| Syntax | Supported |
|--------------------------------|-----------|
| CREATE TABLE | Yes |
| CREATE TABLE LIKE | Yes |
| DROP TABLE | Yes |
| INSERT | Yes |
| COPY | Yes |
| SELECT | Yes |
| TRUNCATE | Yes |
| EXPLAIN | Yes |
| ANALYZE | Yes |
| VACUUM | Yes |
| ALTER TABLE DROP PARTITION | Yes |
| ALTER TABLE ADD PARTITION | Yes |
| ALTER TABLE SET WITH OPTION | Yes |
| ALTER TABLE DROP COLUMN | Yes |
| ALTER TABLE ADD COLUMN | Yes |
| ALTER TABLE ADD NODELIST | Yes |
| ALTER TABLE CHANGE OWNER | Yes |
| ALTER TABLE RENAME COLUMN | Yes |
| ALTER TABLE TRUNCATE PARTITION | Yes |
| CREATE INDEX | Yes |
| DROP INDEX | Yes |
| DELETE | Yes |
| Other ALTER TABLE syntax | Yes |
| ALTER INDEX | Yes |
| MERGE | Yes |
| SELECT INTO | Yes |

| Syntax | Supported |
|-----------------|-----------|
| UPDATE | Yes |
| CREATE TABLE AS | Yes |

Constraints

1. To use HStore tables, use the following parameter settings, or the performance of HStore tables will deteriorate significantly:
autovacuum_max_workers_hstore=3, autovacuum_max_workers=6, and autovacuum=true
2. In version 8.2.1 and later, you can now clear the dirty data from column-store indexes. This is especially beneficial when dealing with frequent data updates and imports into the database. By efficiently managing the index space, it improves both the import and query performance.
3. When using HStore asynchronous sorting, pay attention to the following:
 - DML operations on certain data may be blocked during asynchronous sorting. The maximum blocking granularity is the row threshold for asynchronous sorting. This function is not recommended for frequent DML operations.
 - Automatic asynchronous sorting and column-store VACUUM cannot be used together. If the autovacuum process meets the conditions for column-store VACUUM, asynchronous sorting is skipped and will wait for the next trigger. In some cases, column-store VACUUM may be continuously triggered due to a high volume of DML operations, which means asynchronous sorting will never be triggered.

17.3 Hybrid Data Warehouse Syntax

17.3.1 CREATE TABLE

Function

Create an HStore table in the current database. The table will be owned by the user who created it.

In a hybrid data warehouse, you can use DDL statements to create HStore tables. To create an HStore table, set **enable_hstore** to **on** and set **orientation** to **column**.

To enhance performance, DWS 9.1.0 and later versions have optimized HStore tables and kept the old ones for compatibility purposes. The optimized tables are known as HStore Opt tables. HStore tables can be replaced by HStore Opt tables for better performance, except in scenarios requiring high performance without micro-batch updates.

You cannot convert a created HStore table to an HStore_opt table. Create an HStore Opt table if you want to use it.

Precautions

- When creating an HStore table, ensure that the database GUC parameter settings meet the following requirements:
 - **autovacuum** is set to **on**.
 - The value of **autovacuum_max_workers_hstore** is greater than **0**.
 - The value of **autovacuum_max_workers** is greater than that of **autovacuum_max_workers_hstore**.
- To create an HStore table, you must have the **USAGE** permission on schema cstore.
- The table-level parameters **enable_delta** and **enable_hstore_opt** cannot be enabled at the same time. The parameter **enable_delta** is used to enable delta for common column-store tables and conflicts with **enable_hstore_opt**.
- Each HStore table is bound to a delta table. The OID of the delta table is recorded in the **reldeltaidx** field in **pg_class**. (The **reldelta** field is used by the delta table of the column-store table).

Syntax

```
CREATE TABLE [ IF NOT EXISTS ] table_name
({ column_name data_type
  | LIKE source_table [like_option [...] ] }
)
[, ... ]
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ TABLESPACE tablespace_name ]
[ DISTRIBUTE BY HASH ( column_name [,...] ) ]
[ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
[ PARTITION BY {
  {RANGE (partition_key) ( partition_less_than_item [, ... ] ) }
} [ { ENABLE | DISABLE } ROW MOVEMENT ] ];
```

The options for LIKE are as follows:
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS | PARTITION
| REOPTIONS | DISTRIBUTION | ALL }

Differences Between Delta Tables

Table 17-4 Differences between the delta tables of HStore and column-store tables

| Type | Column-Store Delta Table | HStore Delta Table | HStore Opt Delta Table |
|-----------------|--|--|--|
| Table structure | Same as that defined for the column-store primary table. | Different from that defined for the primary table. | Different from the definitions of the primary table and but same as the definitions of the HStore table. |

| Type | Column-Store Delta Table | HStore Delta Table | HStore Opt Delta Table |
|---------------|--|--|--|
| Functionality | Used to temporarily store a small batch of inserted data. After the data size reaches the threshold, the data will be merged to the primary table. In this way, data will not be directly inserted to the primary table or generate a large number of small CUs. | Persistently stores UPDATE, DELETE, and INSERT information. It is used to restore the memory structure that manages concurrent updates, such as the memory update chain, in the case of a fault. | Persistently stores UPDATE, DELETE, and INSERT information. It is used to restore the memory structure that manages concurrent updates, such as the memory update chain, in the case of a fault. It is further optimized compared with HStore. |
| Weakness | If data is not merged in a timely manner, the delta table will grow large and affect query performance. In addition, the table cannot solve lock conflicts during concurrent updates. | The merge operation depends on the background AUTOVACUUM. | The merge operation depends on the background AUTOVACUUM. |

| Type | Column-Store Delta Table | HStore Delta Table | HStore Opt Delta Table |
|---------------------------|---|--|---|
| Specification differences | Concurrent requests in the same CU are not supported. It is applicable to the scenario where there are not many concurrent updates. | <ol style="list-style-type: none">1. Insertion and update restrictions:<ul style="list-style-type: none">• MERGE INTO does not support concurrent updates of the same row or repeated updates of the same key.• Concurrent UPDATE or DELETE operations on the same row are not supported. Otherwise, an error is reported.2. Index and query restrictions:<ul style="list-style-type: none">• Indexes do not support array condition filtering, IN expression filtering, partial indexes, or expression indexes.• Indexes cannot be invalidated.3. Table structure and operation restrictions:<ul style="list-style-type: none">• Ensure that the tables to be exchanged are HStore tables during partition exchange or relfilenode operations.• The distribution column cannot be modified | <ol style="list-style-type: none">1. Insertion and update restrictions:<ul style="list-style-type: none">• MERGE INTO does not support concurrent updates of the same row or repeated updates of the same key.• Concurrent updates or deletions of the same row is not supported.• hstore_opt does not support cross-partition upserts.2. Index and query restrictions:<ul style="list-style-type: none">• Bitmap indexes are supported.• Global dictionaries are supported.• bitmap_columns must be specified during table creation and cannot be modified after being set.• The opt version does not support transparent parameter transmission during SMP streaming. In multi-table join queries that require partition pruning, avoid using replicated tables or setting query_dop. |

| Type | Column-Store Delta Table | HStore Delta Table | HStore Opt Delta Table |
|-------------------------|---|---|---|
| | | using the UPDATE command. You are not advised to modify the partition column using the UPDATE command. (No error is reported, but the performance is poor.) | 3. Table structure and operation restrictions: <ul style="list-style-type: none">• Distribution columns and partition columns cannot be modified using UPDATE.• The enable_hstore_opt attribute must be set when the table is created and cannot be changed after being set. |
| Data import suggestions | <ol style="list-style-type: none">1. For optimal data import, query performance, and space utilization, it is recommended to choose the HStore Opt table. In scenarios involving micro-batch copying with high performance demands and no data updates, you can choose the HStore table.2. Similarities between HStore and HStore Opt tables:<ul style="list-style-type: none">• The performance of importing data using UPDATE is poor. You are advised to use UPSERT to import data.• When using DELETE to import data, use index scanning. The JDBC batch method is recommended.• Use MERGE INTO to import data records to the database when the data volume exceeds 1 million per DN and there is no concurrent data.• Do not modify or add data in cold partitions. | | |

| Type | Column-Store Delta Table | HStore Delta Table | HStore Opt Delta Table |
|-------------------------|---|--------------------|------------------------|
| Point query suggestions | <ol style="list-style-type: none">1. Generally, the HStore Opt table is recommended for point queries.2. Similarities between HStore and HStore Opt tables:
Create a level-2 partition on the column where the equal-value filter condition is most frequently used and distinct values are evenly distributed.3. Suggestions on using HStore tables for point queries:<ul style="list-style-type: none">• Accelerating indexes other than primary keys may have poor effect. You are advised not to enable index acceleration.• If the data type is numeric or strings less than 16 bytes, Turbo acceleration is recommended.4. Suggestions on using HStore Opt tables:<ul style="list-style-type: none">• For equal-value filter columns not in level-2 partitions, if the columns involved in the filter criteria are basically fixed in the query, use the CB-tree index. If the columns change continuously, you are advised to use the GIN index. Do not select more than five index columns.• For all string columns involving equivalent filtering, bitmap indexes can be specified during table creation. The number of columns is not limited, but cannot be modified later.• Specify columns that can be filtered by time range as the partition columns.• If the number of returned data records exceeds 100,000 per DN, index scanning may not significantly enhance performance. In this case, you are advised to use the GUC parameter enable_seqscan to test the performance then determine which optimization method to use. | | |

Parameters

- **IF NOT EXISTS**
If **IF NOT EXISTS** is specified, a table will be created if there is no table using the specified name. If there is already a table using the specified name, no error will be reported. A message will be displayed indicating that the table already exists, and the database will skip table creation.
- **table_name**
Specifies the name of the table to be created.
The table name can contain a maximum of 63 characters, including letters, digits, underscores (_), dollar signs (\$), and number signs (#). It must start with a letter or underscore (_).
- **column_name**
Specifies the name of a column to be created in the new table.
The column name can contain a maximum of 63 characters, including letters, digits, underscores (_), dollar signs (\$), and number signs (#). It must start with a letter or underscore (_).

- **data_type**
Specifies the data type of the column.
- **LIKE source_table [like_option ...]**
Specifies a table from which the new table automatically copies all column names and their data types.
The new table and the original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and scans on the original table will not be performed on the data of the new table.
Columns copied by **LIKE** are not merged with the same name. If the same name is specified explicitly or in another **LIKE** clause, an error will be reported.
HStore tables can be inherited only from HStore tables.
- **WITH ({ storage_parameter = value } [, ...])**
Specifies an optional storage parameter for a table.
 - **ORIENTATION**
Specifies the storage mode (time series, row-store, or column-store) of table data. This parameter cannot be modified once it is set. For HStore tables, use the column storage mode and set **enable_hstore** to **on**.
Options:
 - **TIMESERIES** indicates that the data is stored in time series.
 - **COLUMN** indicates that the data is stored in columns.
 - **ROW** indicates that table data is stored in rows.Default value: **ROW**
 - **COMPRESSION**
Specifies the compression level of the table data. It determines the compression ratio and time. Generally, a higher compression level indicates a higher compression ratio and a longer compression time, and vice versa. The actual compression ratio depends on the distribution characteristics of loading table data.
Options:
This parameter is available only for HStore tables and column-store tables. Value: **LOW** (default value), **MIDDLE**, or **HIGH**.
 - **COMPRESSLEVEL**
Specifies table data compression rate and duration at the same compression level. This divides a compression level into sub-levels, providing you with more choices for compression ratio and duration. As the value becomes greater, the compression rate becomes higher and duration longer at the same compression level. The parameter is only valid for time series tables and column-store tables.
Value range: 0 to 3
Default value: **0**
 - **MAX_BATCHROW**

Specifies the maximum number of rows in a storage unit during data loading. The parameter is only valid for time series tables and column-store tables.

Value range: 10000 to 60000

Default value: **60000**

– PARTIAL_CLUSTER_ROWS

Specifies the number of records to be partially clustered for storage during data loading. The parameter is only valid for time series tables and column-store tables.

Value range: 600000 to 2147483647

Default value: **4,200,000**

– enable_delta

Specifies whether to enable delta tables in column-store tables. This parameter cannot be enabled for HStore tables.

Default value: **off**

– enable_hstore

Specifies whether to create a table as an HStore table (based on column-store tables). The parameter is only valid for column-store tables. This parameter is supported by version 8.2.0.100 or later clusters.

Default value: **off**

 NOTE

If this parameter is enabled, the following GUC parameters must be set to ensure that HStore tables are cleared.

**autovacuum=true, autovacuum_max_workers=6,
autovacuum_max_workers_hstore=3.**

– enable_disaster_cstore

Specifies whether fine-grained DR will be enabled for column-store tables. This parameter only takes effect on column-store tables whose COLVERSION is 2.0 and cannot be set to **on** if **enable_hstore** is **on**. This parameter is supported by version 8.2.0.100 or later clusters.

Default value: **off**

 CAUTION

Before enabling this function, set the GUC parameter **enable_metadata_tracking** to **on**. Otherwise, fine-grained DR may fail to be enabled.

– SUB_PARTITION_COUNT

Specifies the number of level-2 partitions. This parameter specifies the number of level-2 partitions during data import. This parameter is configured during table creation and cannot be modified after table creation. You are not advised to set the default value, which may affect the import and query performance.

Value range: 1 to 1024

Default value: **32**

- DELTAROW_THRESHOLD
Specifies the maximum number of rows (**SUB_PARTITION_COUNT** x **DELTAROW_THRESHOLD**) to be imported to the delta table.
Value range: 0 to 60000
Default value: **60000**
- COLVERSION
Specifies the version of the storage format. HStore tables support only version 2.0, and **enable_hstore_opt** tables support versions 2.0 and 3.0.
Options:
1.0: Each column in a column-store table is stored in a separate file. The file name is **relfilenode.C1.0**, **relfilenode.C2.0**, **relfilenode.C3.0**, or similar.
2.0: All columns of a column-store table are combined and stored in a file. The file is named **relfilenode.C1.0**.
Default value: **2.0**
- enable_binlog
Specifies whether to enable the binlog function for the HStore Opt table. This parameter is supported only by clusters of version 9.1.0 or later.
Value range: **on** and **off**
Default value: **off**
- enable_binlog_timestamp
Determines whether to enable the binlog function with timestamps for HStore Opt tables. This parameter and **enable_binlog** cannot be enabled at the same time. Only clusters of 9.1.0.200 and later versions support this parameter.
Value range: **on** and **off**
Default value: **off**
- DISTRIBUTE BY
Specifies how the table is distributed or replicated between DN.
Options:
HASH (column_name): Each row of the table will be placed into all the DN based on the hash value of the specified column.
- TO { GROUP groupname | NODE (nodename [, ...]) }
TO GROUP specifies the Node Group in which the table is created. Currently, it cannot be used for HDFS tables. **TO NODE** is used for internal scale-out tools.
- PARTITION BY
Specifies the initial partition of an HStore table.
- secondary_part_column
Specifies the name of a level-2 partition column in a column-store table. Only one column can be specified as the level-2 partition column. This parameter applies only to HStore Opt column-store tables. This parameter is supported only by clusters of version 9.1.0 and later. V3 tables do not support this parameter and will use hashbucket pruning.

 NOTE

- The column specified as a level-2 partition column cannot be deleted or modified.
- The level-2 partition column can be specified only when a table is created. After a table is created, the level-2 partition column cannot be modified.
- You are not advised to specify a distribution column as a level-2 partition column.
- The level-2 partition column determines how the table is logically split into hash partitions on DNs, which enhances the query performance for that column.

– secondary_part_num

Specifies the number of level-2 partitions in a column-store table. This parameter applies only to HStore Opt column-store tables. This parameter is supported only by clusters of version 9.1.0 and later. V3 tables do not support this parameter and will use hashbucket pruning.

Value range: 1 to 32

Default value: 8

 NOTE

- This parameter can be specified only when **secondary_part_column** is specified.
- The number of level-2 partitions can be specified only when a table is created and cannot be modified after the table is created.
- You are not advised to change the default value, which may affect the import and query performance.

Example

Create a simple HStore Opt table.

```
CREATE TABLE warehouse_t1
(
  W_WAREHOUSE_SK      INTEGER      NOT NULL,
  W_WAREHOUSE_ID      CHAR(16)     NOT NULL,
  W_WAREHOUSE_NAME     VARCHAR(20) ,
  W_WAREHOUSE_SQ_FT   INTEGER      ,
  W_STREET_NUMBER     CHAR(10)     ,
  W_STREET_NAME       VARCHAR(60)  ,
  W_STREET_TYPE       CHAR(15)     ,
  W_SUITE_NUMBER      CHAR(10)     ,
  W_CITY              VARCHAR(60)   ,
  W_COUNTY            VARCHAR(30)   ,
  W_STATE             CHAR(2)       ,
  W_ZIP              CHAR(10)      ,
  W_COUNTRY           VARCHAR(20)   ,
  W_GMT_OFFSET        DECIMAL(5,2)
)WITH(ORIENTATION=COLUMN, ENABLE_HSTORE_OPT=ON);

CREATE TABLE warehouse_t2 (LIKE warehouse_t1 INCLUDING ALL);
```

17.3.2 INSERT

Function

Insert one or more rows of data into an HStore table.

Precautions

- If the data to be inserted at a time is greater than or equal to the value of the table-level parameter **DELTA_ROW_THRESHOLD**, the data is directly inserted into the primary table to generate a compression unit (CU).
- If the data to be inserted is smaller than **DELTA_ROW_THRESHOLD**, a record of the type I will be inserted into the delta table. The data will be serialized and stored in the **values** field of the record.
- CUIDs are allocated to the data in the delta table and the primary table in a unified manner.
- The data inserted into the delta table depends on **AUTOVACUUM** to merge to primary table CUs.

Syntax

```
INSERT [/*+ plan_hint */] [ IGNORE | OVERWRITE ] INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]  
{ DEFAULT VALUES  
| VALUES {( { expression | DEFAULT } [, ...] ) }, ... ] | query }
```

Parameters

- **table_name**
Specifies the name of the target table.
Value range: an existing table name
- **AS**
Specifies an alias for the target table *table_name*. *alias* indicates the alias name.
- **column_name**
Specifies the name of a column in a table.
- **query**
Specifies a query statement (**SELECT** statement) that uses the query result as the inserted data.

Example

Create the **reason_t1** table.

```
-- Create the reason_t1 table.  
CREATE TABLE reason_t1  
(  
    TABLE_SK      INTEGER          ,  
    TABLE_ID      VARCHAR(20)      ,  
    TABLE_NAME     VARCHAR(20)      ,  
)WITH(ORIENTATION=COLUMN, ENABLE_HSTORE_OPT=ON);
```

Insert a record into a table.

```
INSERT INTO reason_t1(TABLE_SK, TABLE_ID, TABLE_NAME) VALUES (1, 'S01', 'StudentA');
```

Insert records into the table.

```
INSERT INTO reason_t1 VALUES (1, 'S01', 'StudentA'),(2, 'T01', 'TeacherA'),(3, 'T02', 'TeacherB');  
SELECT * FROM reason_t1 ORDER BY 1;  
TABLE_SK | TABLE_ID | TABLE_NAME  
-----+-----+-----  
1 | S01 | StudentA  
2 | T01 | TeacherA  
3 | T02 | TeacherB  
(3 rows)
```


17.3.3 DELETE

Function

Delete data from an HStore Opt table.

Precautions

- To delete all the data from a table, you are advised to use the **TRUNCATE** syntax to improve performance and reduce table bloating.
- If a single record is deleted from an HStore Opt table, a record of the type **D** will be inserted into the delta table. The memory update chain will also be updated to manage concurrency.
- If multiple records are deleted from an HStore Opt table at a time, a record of the type **MD** will be inserted for the consecutive deleted records in each CU.
- In concurrent deletion scenarios, operations on the same CU will get queued in traditional column-store tables and result in low performance. For HStore Opt tables, the operations can be concurrently performed, and the DELETE performance can be more than 100 times that of column-store tables.

Syntax

```
DELETE FROM [ ONLY ] table_name [ * ] [ [ AS ] alias ]  
[ USING using_list ]  
[ WHERE condition ]
```

Parameters

- **ONLY**
If **ONLY** is specified, only that table is deleted. If **ONLY** is not specified, this table and all its sub-tables are deleted.
- **table_name**
Specifies the name (optionally schema-qualified) of a target table.
Value range: an existing table name
- **alias**
Specifies the alias for the target table.
Value range: a string. It must comply with the naming convention.
- **using_list**
Specifies the **USING** clause.
- **condition**
Specifies an expression that returns a value of type boolean. Only rows for which this expression returns **true** will be deleted.

Example

Create the **reason_t2** table.

```
CREATE TABLE reason_t2  
(  
  TABLE_SK    INTEGER      ,  
  TABLE_ID    VARCHAR(20)  ,  
  TABLE_NA    VARCHAR(20)
```

```
)WITH(ORIENTATION=COLUMN, ENABLE_HSTORE_OPT=ON);  
INSERT INTO reason_t2 VALUES (1, 'S01', 'StudentA'),(2, 'T01', 'TeacherA'),(3, 'T02', 'TeacherB');
```

Use the **WHERE** condition for deletion.

```
DELETE FROM reason_t2 WHERE TABLE_SK = 2;  
DELETE FROM reason_t2 AS rt2 WHERE rt2.TABLE_SK = 2;
```

Use the **IN** syntax for deletion.

```
DELETE FROM reason_t2 WHERE TABLE_SK in (1,3);
```

17.3.4 UPDATE

Function

Update specified data in an HStore Opt table.

Precautions

- Similar to column storage, the UPDATE operation on an HStore Opt table in the current version involves DELETE and INSERT. You can configure a global GUC parameter to control the lightweight UPDATE of HStore Opt. In the current version, the lightweight UPDATE is disabled by default.
- In concurrent update scenarios, operations on the same CU will cause lock conflicts in traditional column-store tables and result in low performance. For HStore Opt tables, the operations can be concurrently performed, and the update performance can be more than 100 times that of column-store tables.

Syntax

```
UPDATE [/*+ plan_hint */] [ ONLY ] table_name [ * ] [ [ AS ] alias ]  
SET {column_name = { expression | DEFAULT }  
    | ( column_name [, ...] ) = { ( { expression | DEFAULT } [, ...] ) |sub_query }}, ...]  
[ FROM from_list ] [ WHERE condition ];
```

Parameters

- **plan_hint** clause
Following the keyword in the */*+ */* format, hints are used to optimize the plan generated by a specified statement block. For details, see "Performance Tuning" > "Query Improvement" > "Hint-based Tuning" in *Data Warehouse Service (DWS) Developer Guide*.
- **table_name**
Name (optionally schema-qualified) of the table to be updated.
Value range: an existing table name
- **alias**
Specifies the alias for the target table.
Value range: a string. It must comply with the naming convention.
- **expression**
Specifies a value assigned to a column or an expression that assigns the value.
- **DEFAULT**
Sets the column to its default value.
The value is **NULL** if no specified default value has been assigned to it.

- **from_list**

A list of table expressions, allowing columns from other tables to appear in the **WHERE** condition and the update expressions. This is similar to the list of tables that can be specified in the **FROM** clause of a **SELECT** statement.

NOTICE

Note that the target table must not appear in the **from_list**, unless you intend a self-join (in which case it must appear with an alias in the **from_list**).

- **condition**

An expression that returns a value of type **boolean**. Only rows for which this expression returns **true** are updated.

Example

Create the **reason_update** table.

```
CREATE TABLE reason_update
(
  TABLE_SK      INTEGER      ,
  TABLE_ID      VARCHAR(20)  ,
  TABLE_NA      VARCHAR(20)
)WITH(ORIENTATION=COLUMN, ENABLE_HSTORE_OPT=ON);
```

Insert data to the **reason_update** table.

```
INSERT INTO reason_update VALUES (1, 'S01', 'StudentA'),(2, 'T01', 'TeacherA'),(3, 'T02', 'TeacherB');
```

Perform the UPDATE operation on the **reason_update** table.

```
UPDATE reason_update SET TABLE_NA = 'TeacherD' where TABLE_SK = 3;
```

17.3.5 UPSERT

Function

HStore Opt is compatible with the **UPSERT** syntax. You can add one or more rows to a table. When a row duplicates an existing primary key or unique key value, the row will be ignored or updated.

Precautions

- The **UPSERT** statement of updating data upon conflict can be executed only when the target table contains a primary key or unique index.
- Similar to column storage, an update operation performed using **UPSERT** on an HStore Opt table in the current version involves DELETE and INSERT.
- In concurrent **UPSERT** scenarios, operations on the same CU will cause lock conflicts in traditional column-store tables and result in low performance. For HStore Opt tables, the operations can be concurrently performed, and the upsert performance can be more than 100 times that of column-store tables.

Syntax

Table 17-5 UPSERT syntax

| Syntax | Update Data Upon Conflict | Ignore Data Upon Conflict |
|---|--|--|
| Syntax 1:
No index is specified. | INSERT INTO ON DUPLICATE KEY UPDATE | INSERT IGNORE
INSERT INTO ON CONFLICT DO NOTHING |
| Syntax 2:
The unique key constraint can be inferred from the specified column name or constraint name. | INSERT INTO ON CONFLICT(...) DO UPDATE SET
INSERT INTO ON CONFLICT ON CONSTRAINT con_name DO UPDATE SET | INSERT INTO ON CONFLICT(...) DO NOTHING
INSERT INTO ON CONFLICT ON CONSTRAINT con_name DO NOTHING |

Parameters

In syntax 1, no index is specified. The system checks for conflicts on all primary keys or unique indexes. If a conflict exists, the system ignores or updates the corresponding data.

In syntax 2, a specified index is used for conflict check. The primary key or unique index is inferred from the column name, the expression that contains column names, or the constraint name specified in the **ON CONFLICT** clause.

- **Unique index inference**
Syntax 2 infers the primary key or unique index by specifying the column name or constraint name. You can specify a single column name or multiple column names by using an expression. Example: **column1, column2, column3**
- **UPDATE clause**
The **UPDATE** clause can use **VALUES(colname)** or **EXCLUDED.colname** to reference inserted data. **EXCLUDED** indicates the rows that should be excluded due to conflicts.
- **WHERE clause**
 - The **WHERE** clause is used to determine whether a specified condition is met when data conflict occurs. If yes, update the conflict data. Otherwise, ignore it.
 - Only syntax 2 of **Update Data Upon Conflict** can specify the **WHERE** clause, that is, **INSERT INTO ON CONFLICT(...) DO UPDATE SET WHERE**.

Example

Create table **reason_upsert** and insert data into it.

```
CREATE TABLE reason_upsert
(
  a int primary key,
  b int,
  c int
)WITH(ORIENTATION=COLUMN, ENABLE_HSTORE_OPT=ON);
INSERT INTO reason_upsert VALUES (1, 2, 3);
```

Ignore conflicting data.

```
INSERT INTO reason_upsert VALUES (1, 4, 5),(2, 6, 7) ON CONFLICT(a) DO NOTHING;
```

Update conflicting data.

```
INSERT INTO reason_upsert VALUES (1, 4, 5),(3, 8, 9) ON CONFLICT(a) DO UPDATE SET b = EXCLUDED.b,
c = EXCLUDED.c;
```

17.3.6 MERGE INTO

Function

The **MERGE INTO** statement is used to conditionally match data in a target table with that in a source table. If data matches, **UPDATE** is executed on the target table; if data does not match, **INSERT** is executed. You can use this syntax to run **UPDATE** and **INSERT** at a time for convenience.

Precautions

In concurrent **MERGE INTO** scenarios, the update operations triggered on the same CU will cause lock conflicts in traditional column-store tables and result in low performance. For HStore Opt tables, the operations can be concurrently performed, and the **MERGE INTO** performance can be more than 100 times that of column-store tables.

Syntax

```
MERGE INTO table_name [ [ AS ] alias ]
USING { { table_name | view_name } | subquery } [ [ AS ] alias ]
ON ( condition )
[
  WHEN MATCHED THEN
    UPDATE SET { column_name = { expression | DEFAULT } |
      ( column_name [, ...] ) = ( { expression | DEFAULT } [, ...] ) } [, ...]
  [ WHERE condition ]
]
[
  WHEN NOT MATCHED THEN
    INSERT { DEFAULT VALUES |
      [ ( column_name [, ...] ) ] VALUES ( { expression | DEFAULT } [, ...] ) [, ...] [ WHERE condition ] }
];
```

Parameters

- **INTO** clause
Specifies the target table that is being updated or has data being inserted.
 - **table_name**
Specifies the name of the target table.

- **alias**
Specifies the alias for the target table.
Value range: a string. It must comply with the naming convention.
- **USING** clause
Specifies the source table, which can be a table, view, or subquery.
- **ON** clause
Specifies the condition used to match data between the source and target tables. Columns in the condition cannot be updated. The **ON** association condition can be **ctid**, **xc_node_id**, or **tableoid**.
- **WHEN MATCHED** clause
Performs **UPDATE** if data in the source table matches that in the target table based on the condition.

 NOTE

Distribution columns, system catalogs, and system columns cannot be updated.

- **WHEN NOT MATCHED** clause
Specifies that the **INSERT** operation is performed if data in the source table does not match that in the target table based on the condition.

 NOTE

- An **INSERT** clause can contain only one **VALUES**.
- The sequence of **WHEN NOT MATCHED** and **WHEN NOT MATCHED** clauses can be exchanged. One of them can be omitted, but they cannot be omitted at the same time.
- Two **WHEN MATCHED** or **WHEN NOT MATCHED** clauses cannot be specified at the same time.

Example

Create a target for **MERGE INTO**.

```
CREATE TABLE target(a int, b int)WITH(ORIENTATION = COLUMN, ENABLE_HSTORE_OPT = ON);
INSERT INTO target VALUES(1, 1),(2, 2);
```

Create a data source table.

```
CREATE TABLE source(a int, b int)WITH(ORIENTATION = COLUMN, ENABLE_HSTORE_OPT = ON);
INSERT INTO source VALUES(1, 1),(2, 2),(3, 3),(4, 4),(5, 5);
```

Run the **MERGE INTO** command.

```
MERGE INTO target t
USING source s
ON (t.a = s.a)
WHEN MATCHED THEN
    UPDATE SET t.b = t.b + 1
WHEN NOT MATCHED THEN
    INSERT VALUES (s.a, s.b) WHERE s.b % 2 = 0;
```

17.3.7 SELECT

Function

Reads data from an HStore Opt table.

Precautions

- Currently, neither column-store tables and HStore Opt tables support the **SELECT FOR UPDATE** syntax.
- When a **SELECT** query is performed on an HStore Opt table, the system will scan the data in column-store primary table CUs, the delta table, and the update information in each row in the memory. The three types of information will be combined before returned.
- In the scenario where data is queried using the primary key index or unique index:

For traditional column-store tables, the unique index stores both the data location information (blocknum, offset) of the row-store Delta table and the data location information (cuid, offset) of the column-store primary table. After the data is merged to the primary table, a new index tuple will be inserted, and the index will keep bloating.

For HStore Opt tables, global CUIDs are allocated in a unified manner. Therefore, only cuid and offset are stored in index tuples. After data is merged, no new index tuples will be generated.

Syntax

```
[ WITH [ RECURSIVE ] with_query [, ...] ]  
SELECT [/*+ plan_hint */] [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]  
{ * | {expression [ [ AS ] output_name ]} [, ...] }  
[ FROM from_item [, ...] ]  
[ WHERE condition ]  
[ GROUP BY grouping_element [, ...] ]  
[ HAVING condition [, ...] ]  
[ { UNION | INTERSECT | EXCEPT | MINUS } [ ALL | DISTINCT ] select ]  
[ ORDER BY {expression [ [ ASC | DESC | USING operator ] | nlssort_expression_clause } [ NULLS { FIRST |  
LAST } ]} [, ...] ]  
[ { [ LIMIT { count | ALL } ] [ OFFSET start [ ROW | ROWS ] ] } | { LIMIT start, { count | ALL } } ]
```

Parameters

- **DISTINCT [ON (expression [, ...])]**
Removes all duplicate rows from the **SELECT** result set.
ON (expression [, ...]) is only reserved for the first row among all the rows with the same result calculated using given expressions.
- **SELECT list**
Indicates columns to be queried. Some or all columns (using wildcard character *) can be queried.
You may use the **AS output_name** clause to give an alias for an output column. The alias is used for the displaying of the output column.
- **FROM clause**
Indicates one or more source tables for **SELECT**.
The **FROM** clause can contain the following elements:
- **WHERE clause**
The **WHERE** clause forms an expression for row selection to narrow down the query range of **SELECT**. The condition is any expression that evaluates to a result of Boolean type. Rows that do not satisfy this condition will be eliminated from the output.

In the **WHERE** clause, you can use the operator (+) to convert a table join to an outer join. However, this method is not recommended because it is not the standard SQL syntax and may raise syntax compatibility issues during platform migration. There are many restrictions on using the operator (+):

- **GROUP BY clause**

Condenses query results into a single row all selected rows that share the same values for the grouped expressions.

- **HAVING clause**

Selects special groups by working with the **GROUP BY** clause. The **HAVING** clause compares some attributes of groups with a constant. Only groups that matching the logical expression in the **HAVING** clause are extracted.

- **ORDER BY clause**

Sorts data retrieved by **SELECT** in descending or ascending order. If the **ORDER BY** expression contains multiple columns:

Example

Create the **reason_select** table and insert data into the table.

```
CREATE TABLE reason_select
(
  r_reason_sk integer,
  r_reason_id integer,
  r_reason_desc character(100)
)WITH(ORIENTATION = COLUMN, ENABLE_HSTORE_OPT=ON);
INSERT INTO reason_select values(3, 1,'reason 1'),(10, 2,'reason 2'),(4, 3,'reason 3'),(10, 4,'reason 4');
```

Perform the **GROUP BY** operation.

```
SELECT COUNT(*), r_reason_sk FROM reason_select GROUP BY r_reason_sk;
```

Perform the **HAVING** filtering operation.

```
SELECT COUNT(*) c,r_reason_sk FROM reason_select GROUP BY r_reason_sk HAVING c > 1;
```

Perform the **ORDER BY** operation.

```
SELECT * FROM reason_select ORDER BY r_reason_sk;
```

17.3.8 ALTER TABLE

Function

Modify a table, including modifying the definition of a table, renaming a table, renaming a specified column in a table, adding or updating multiple columns, and changing a column-store table to an HStore Opt table.

Precautions

- You can set **enable_hstore** by using **ALTER** to change a column-store table to an HStore table, or to change it back. If **enable_delta** is set to **on**, **enable_hstore** cannot be set to **on**.
- For some **ALTER** operations (such as modifying column types, merging partitions, adding NOT NULL constraints, and adding primary key constraints), HStore tables need to merge data to the primary table and then perform **ALTER**, which may cause extra performance overhead. The impact on performance depends on the data volume in the delta table.
- When you add a column, do not use **ALTER** to specify other operations (for example, modifying the column type). An **ALTER** statement with only the

ADD COLUMN parameter can achieve high performance, because it does not require FULL MERGE.

- The storage parameter **ORIENTATION** cannot be modified.

Modifying Table Attributes

Syntax:

```
ALTER TABLE [ IF EXISTS ] <table_name> SET ( {ENABLE_HSTORE = ON} [, ... ] );
```

To change a column-store table to an HStore table, run the following command:

```
CREATE TABLE alter_test(a int, b int) WITH(ORIENTATION = COLUMN);  
ALTER TABLE alter_test SET (ENABLE_HSTORE = ON);
```

NOTICE

To use HStore tables, set the following parameters, or the HStore performance will deteriorate severely. The recommended settings are as follows:

**autovacuum_max_workers_hstore=3, autovacuum_max_workers=6,
autovacuum=true**

Adding a Column

Syntax:

```
ALTER TABLE [ IF EXISTS ] <table_name> ADD COLUMN <new_column> <data_type> [ DEFAULT  
<default_value>];
```

Example:

Create the **alter_test2** table and add a column to it.

```
CREATE TABLE alter_test2(a int, b int) WITH(ORIENTATION = COLUMN,ENABLE_HSTORE_OPT = ON);  
ALTER TABLE alter_test ADD COLUMN c int;
```

NOTE

When adding a column, you are not advised to use **ALTER** to specify other operations in the same SQL statement.

Renaming

Syntax:

```
ALTER TABLE [ IF EXISTS ] <table_name> RENAME TO <new_table_name>;
```

Example:

Create table **alter_test3** and rename it as **alter_new**.

```
CREATE TABLE alter_test3(a int, b int) WITH(ORIENTATION = COLUMN,ENABLE_HSTORE_OPT = ON);  
ALTER TABLE alter_test3 RENAME TO alter_new;
```

17.4 Hybrid Data Warehouse Functions

hstore_light_merge(rel_name text)

Description: Allows users to manually perform lightweight cleanup on HStore tables and holds the level-3 lock of the target table.

Return type: int

Example:

```
SELECT hstore_light_merge('reason_select');
```

hstore_full_merge(rel_name text, partitionName text)

Description: Allows users to manually perform full cleanup on HStore tables.

partitionName is an optional parameter and is used to specify a single partition for operations.

Return type: int

NOTE

- This operation forcibly merges all the visible operations of the delta table to the primary table, and then creates an empty delta table. During this period, this operation holds the level-8 lock of the table.
- The duration of this operation depends on the amount of data in the delta table. You must enable the HStore clearing thread to ensure unnecessary data in the HStore table is cleared in a timely manner.
- The second parameter **partitionName** is only supported by clusters of version 9.1.0 or later. However, these versions do not allow calling this function via **call** because it lacks reload capability.

Example:

```
SELECT hstore_full_merge('reason_select', 'part1');
```

pgxc_get_small_cu_info(rel_name text, row_count int)

Description: Obtains the small CU information of the target table. This function is supported only by clusters of version 8.2.1.200 or later.

row_count is an optional parameter, which indicates the threshold of small CUs. If the number of live tuples is less than the value of this parameter, small CUs are used. The default value is **200**.

Return type: record

Return value:

- **node_name:** DN name.
- **part_name:** partition name. This column is empty for non-partitioned tables.
- **zero_cu_count:** number of 0 CUs. If all data in a CU is deleted, the CU is called 0 CU.

- **small_cu_count**: number of small CUs. When a CU has live data that is less than the threshold, the CU is called a small CU.
- **total_cu_count**: total number of CUs.
- **sec_part_cu_num**: number of CUs in each level-2 partition. This column is displayed only when **secondary_part_column** is specified. This field is available only in clusters of version 8.3.0 or later.

 NOTE

A CU may contain multiple columns.

Example:

```
SELECT * FROM pgxc_get_small_cu_info('hs');
node_name | part_name | zero_cu_count | small_cu_count | total_cu_count |
sec_part_cu_num
-----+-----+-----+-----+-----+-----
datanode1 |          | 0 | 4 | 4 | p1:1 p2:0 p3:1 p4:0 p5:1 p6:0 p7:1 p8:0
datanode2 |          | 0 | 4 | 4 | p1:0 p2:1 p3:0 p4:1 p5:0 p6:1 p7:0 p8:1
(2 rows)
```

gs_hstore_compaction(rel_name text, row_count int)

Description: Merges small CUs of the target table. This function is supported only by clusters of version 8.2.1.200 or later.

row_count is an optional parameter, which indicates the threshold of small CUs. If the number of live tuples is less than the value of this parameter, small CUs are used. The default value is **100**.

Return type: int

Return value: **numCompactCU**, which indicates the number of small CUs to be merged.

 NOTE

- A CU may contain multiple columns.
- You cannot specify a partition name for this function. That is, this function does not support a single partition.

Example:

```
SELECT gs_hstore_compaction('hs', 10);
```

pgxc_get_hstore_delta_info(rel_name text)

Description: Obtains the delta table information of the target table, including the delta table size and the number of **INSERT**, **DELETE**, and **UPDATE** records. This function is supported only by clusters of version 8.2.1.100 or later.

Return type: record

Return value:

- **node_name**: DN name.
- **part_name**: partition name. This column is set to **non-partition table** if the table is not a partitioned table.

- **live_tup**: number of live tuples.
- **n_ui_type**: number of records with a type of *ui* (small CU combination and upsert insertion through update). An **ui** record represents a single or batch insertion. This parameter is supported only by 8.3.0.100 and later versions.
- **n_i_type**: number of records whose type is **i** (insert). An **i** record indicates one insertion, which can be single insertion or batch insertion.
- **n_d_type**: number of records whose type is **d** (delete). One **d** record indicates one deletion, which can be single deletion or batch deletion.
- **n_x_type**: number of records whose type is **x** (deletions generated by update).
- **n_u_type**: number of records whose type is **u** (lightweight update).
- **n_m_type**: number of records whose type is **m** (merge).
- **data_size**: total size of the **delta** table, in bytes (including the size of the index and **toast** data on the **delta** table).

Example:

```
SELECT * FROM pgxc_get_hstore_delta_info('hs_part');
node_name | part_name | live_tup | n_ui_type | n_i_type | n_d_type | n_x_type | n_u_type | n_m_type | data_size
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
dn_1      | p1        | 2        | 0         | 2        | 0        | 0        | 0        | 0        | 8192
dn_1      | p2        | 2        | 0         | 2        | 0        | 0        | 0        | 0        | 8192
dn_1      | p3        | 2        | 0         | 2        | 0        | 0        | 0        | 0        | 8192
(3 rows)
```

pgxc_get_binlog_sync_point(rel_name text, slot_name text, checkpoint bool, node_id int)

Description: Obtains the synchronization point information corresponding to a slot from the **pg_binlog_slots** system catalog. This function is applicable only to tables with binlog or binlog timestamp enabled. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: record

Return value:

- **node_name**: node name
- **node_id**: node ID
- **last_sync_point**: last synchronization point
- **latest_sync_point**: latest synchronization point
- **xmin**: xmin corresponding to the synchronization point

Example:

```
SELECT * FROM pg_catalog.pgxc_get_binlog_sync_point('hstore_binlog_source', 'slot1', false, 0);
node_name | node_id | last_sync_point | latest_sync_point | xmin
-----+-----+-----+-----+-----
dn_2      | -1051926843 | 0              | 10512             | 10507
dn_1      | -1300059100 | 0              | 10512             | 10508
(2 rows)
```

pgxc_get_binlog_changes(rel_name text, node_id int, start_csn bigint, end_csn bigint)

Description: Obtains the incremental data of the target table within the specified synchronization point range on a specified DN. If **node_id** is set to **0**, all DNs are

specified. This function is applicable only to tables with binlog or binlog timestamp enabled. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: record

Return value:

- **gs_binlog_sync_point:** synchronization point
- **gs_binlog_event_sequence:** sequence in the same transaction
- **gs_binlog_event_type:** binlog type
- **gs_binlog_timestamp_us:** timestamp of the binlog record. For the binlog table whose **enable_binlog_timestamp** is **false**, this column is empty.
- **value columns:** data of each user field in the target table

Example:

```
SELECT * FROM pgxc_get_binlog_changes('hstore_binlog_source', 0, 0, 9999999999);
gs_binlog_sync_point | gs_binlog_event_sequence | gs_binlog_event_type | gs_binlog_timestamp_us | c1 | c2 | c3
-----+-----+-----+-----+-----+-----+-----
10516 | 2 | I | 1731570520900211 | 100 | 1 | 1
10517 | 3 | d | 1731570520904425 | 100 | 1 | 1
10518 | 2 | I | 1731570520909055 | 200 | 1 | 1
10519 | 3 | B | 1731570520914102 | 200 | 1 | 1
10519 | 4 | U | 1731570520914154 | 200 | 2 | 1
```

pgxc_register_binlog_sync_point(rel_name text, slot_name text, node_id int, end_csn bigint, checkpoint bool, xmin bigint)

Description: Registers synchronization points and can be used only for tables with binlog or binlog timestamp enabled. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: int

Return value: number of nodes that are successfully registered

Example:

```
SELECT pgxc_register_binlog_sync_point('hstore_binlog_source', 'slot1', 0, 9999999999, false, 100);
pgxc_register_binlog_sync_point
-----
2
(1 row)
```

pgxc_consumed_binlog_records(rel_name text, node_id int)

Description: Obtains the consumption status of the target table on a specified DN. This function can be used only for tables with binlog or binlog timestamp enabled. This function is supported only by clusters of version 9.1.0.200 or later.

If **node_id** is set to **0**, all DNs are queried.

Return type: int

Return value:

- **0:** The binlog of the target table is not completely consumed (including all slots and checkpoint synchronization points).

- **1**: The binlog of the target table is completely consumed.

Example:

```
SELECT * FROM pgxc_consumed_binlog_records('hstore_binlog_source',0);
pgxc_consumed_binlog_records
-----
1
(1 row)
```

pgxc_get_binlog_slots(rel_name text, node_id int)

Description: Obtains the progress information of all slots in the target table. This function is supported only by clusters of version 9.1.0.223 or later.

Return type: record

If **node_id** is set to **0**, all DN's are queried.

Return value:

- **node_name**: node name
- **slot_name**: slot name
- **sync_point**: latest synchronization point registered by the slot
- **sync_timestamp**: last registration time of the slot
- **oldest_xmin**: corresponds to **oldest_xmin** during the latest registration of the slot
- **is_checkpoint**: whether the location is a checkpoint.

Example:

```
SELECT * FROM pgxc_get_binlog_slots('binlog_view_part_t1', 0);
node_name | slot_name | sync_point | sync_timestamp | oldest_xmin | is_checkpoint
-----+-----+-----+-----+-----+-----
dn_1      | slot1     | 210059    | 2025-09-19 10:31:38.894357+08 | 10054      | f
(2 rows)
```

pgxc_binlog_clear_target_slot(rel_name text, slot_name text)

Description: Clears the specified slot of the target table. This function is supported only by clusters of version 9.1.0.223 or later.

Return type: int

Return value: total number of cleared slots on all nodes

Example:

```
SELECT * FROM pgxc_binlog_clear_target_slot('binlog_view_part_t1', 'slot1');
pgxc_binlog_clear_target_slot
-----
4
(1 row)
```

pgxc_get_binlog_cursor_by_timestamp(rel_name text, timestamp timestampTz, node_id int)

Description: Obtains information about the first binlog record after a specified time point in the target table. This function can be used only for tables with the

binlog timestamp enabled. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: record

Return value:

- **node_name:** node name
- **node_id:** node ID
- **latest_sync_point:** latest synchronization point
- **binlog_sync_point:** synchronization point of the first binlog record after the time point
- **binlog_timestamp_us:** timestamp of the first binlog record after the time point
- **binlog_xmin:** xmin recorded in the first binlog after the time point

Example:

```
SELECT * FROM pgxc_get_binlog_cursor_by_timestamp('hstore_binlog_source','2024-11-14
15:48:40.900211+08', 0);
node_name | node_id | latest_sync_point | binlog_sync_point | binlog_timestamp_us | binlog_xmin
-----+-----+-----+-----+-----+-----
dn_2      | -1051926843 | 10532 | 10516 | 1731570520900211 | 10510
dn_1      | -1300059100 | 10532 | 10518 | 1731570520909055 | 10510
(2 rows)
```

pgxc_get_binlog_cursor_by_syncpoint(rel_name text, csn int8, node_id int)

Description: Obtains the first binlog record after a specified synchronization point on the target table. This function can be used only for tables with the binlog timestamp enabled. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: record

Return value:

- **node_name:** node name
- **node_id:** node ID
- **latest_sync_point:** latest synchronization point
- **binlog_sync_point:** synchronization point of the first binlog record after the time point
- **binlog_timestamp_us:** timestamp of the first binlog record after the time point
- **binlog_xmin:** xmin recorded in the first binlog after the time point

Example:

```
SELECT * FROM pgxc_get_binlog_cursor_by_syncpoint('hstore_binlog_source',10516,0);
node_name | node_id | latest_sync_point | binlog_sync_point | binlog_timestamp_us | binlog_xmin
-----+-----+-----+-----+-----+-----
dn_1      | -1300059100 | 11187 | 10518 | 1731570520909055 | 10510
dn_2      | -1051926843 | 11187 | 10516 | 1731570520900211 | 10510
(2 rows)
```

pgxc_get_binlog_consume_progress(rel_name text, node_id int)

Description: Obtains the consumption progress of the target table. This function can be used only for tables with the binlog timestamp enabled. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: record

Return value:

- **node_name:** node name
- **node_id:** node ID
- **slot_name:** slot name
- **checkpoint:** whether the location is a checkpoint
- **latest_consumed_timestamp:** timestamp of the latest consumed binlog
- **latest_timestamp:** latest timestamp in all binlogs
- **latest_consumed_csn:** CSN of the latest consumed binlog
- **latest_csn:** latest CSN in all binlogs
- **unconsumed_binlog_count:** number of binlogs that have not been consumed

Example:

```
SELECT * FROM pgxc_get_binlog_consume_progress('hstore_binlog_source', 0);
 node_name | node_id | slot_name | checkpoint | latest_consumed_timestamp | latest_timestamp |
 latest_consumed_csn | latest_csn | unconsumed_binlog_count
-----+-----+-----+-----+-----+-----+-----
dn_1      | -1300059100 | slot1 | f          | 2025-04-15 15:55:55+08 | 2025-04-15 15:55:55+08 |
417725    | 417725 |          | 0          |                          |                          |
dn_1      | -1300059100 | slot1 | t          | 2025-04-15 15:55:55+08 | 2025-04-15 15:55:55+08 |
417725    | 417725 |          | 0          |                          |                          |
(2 rows)
```

pgxc_get_cstore_dirty_ratio(rel_name text, partition_name)

Description: Obtains the CU, delta, and cudesc dirty page rates and sizes of the target table on each DN. Only HStore Opt tables are supported. It is supported only by clusters of version 9.1.0.100 or later.

partition_name is optional. If the partition name is specified, only the information about the partition is returned. If the partition name is not specified and the table is a primary table, the information about all partitions is returned.

Return type: record

Return value:

- **node_name:** DN name.
- **database_name:** name of the database where the table is located
- **rel_name:** primary table name
- **part_name:** partition name
- **cu_dirty_ratio:** dirty page rate of CU files
- **cu_size:** CU file size
- **delta_dirty_ratio:** dirty page rate of the delta table

- **delta_size**: delta table size
- **cudesc_dirty_ratio**: dirty page rate of the cudesc table
- **cudesc_size**: cudesc table size

Example:

```
SELECT * FROM pgxc_get_cstore_dirty_ratio('hs_opt_part');
node_name | database_name | rel_name | partition_name | cu_dirty_ratio | cu_size | delta_dirty_ratio |
delta_size | cudesc_dirty_ratio | cudesc_size
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
dn_1      | postgres     | public.hs_opt_part | p1             | 0 | 0 | 0 | 16384 |
0 | 24576
dn_1      | postgres     | public.hs_opt_part | p2             | 0 | 0 | 0 | 16384 |
0 | 24576
dn_1      | postgres     | public.hs_opt_part | p3             | 0 | 0 | 0 | 16384 |
0 | 24576
dn_1      | postgres     | public.hs_opt_part | p4             | 0 | 0 | 0 | 16384 |
0 | 24576
dn_1      | postgres     | public.hs_opt_part | other          | 0 | 1105920 | 0 | 524288 |
0 | 40960
```

17.5 Hybrid Data Warehouse Binlog

17.5.1 Subscribing to Hybrid Data Warehouse Binlog

Binlog Usage

The HStore table within the DWS hybrid data warehouse offers binlog to facilitate the capture of database events. This enables the export of incremental data to third-party components like Flink. By consuming binlog data, you can synchronize upstream and downstream data, improving data processing efficiency.

Unlike traditional MySQL binlog, which logs all database changes and focuses on data recovery and replication. The DWS hybrid data warehouse binlog is optimized for real-time data synchronization, recording DML operations—INSERT, DELETE, and UPDATE (UPSERT)—while excluding DDL operations.

DWS Binlog has the following advantages:

- Table-level on-demand switch: enables or disables binlog for specific tables as needed.
- Full incremental integrated consumption: supports full synchronization followed by real-time incremental consumption after a Flink task is started.
- Cleanup upon consumption: allows asynchronous clearing of incremental data after consumption, reducing space usage.

With Flink's real-time processing capabilities and Binlog, you can build a hybrid data warehouse efficiently without additional components like Kafka. The architecture is streamlined, and data flows efficiently, driven by Flink SQL.

Constraints and Limitations

1. Currently, only 8.3.0.100 and later versions support HStore and HStore Opt to record binlogs. V3 tables are in the trial commercial use phase. Before using them, contact technical support for evaluation.

2. Binlog requires a primary key, an HStore or HStore Opt table, and supports only hash distribution.
3. Binlog tables log DML operations like INSERT, DELETE, and UPDATE (UPSERT), excluding DDL operations.
4. Binlog tables do not support INSERT OVERWRITE, altering distribution columns, enabling Binlog on temporary tables, or partition operations like EXCHANGE, MERGE, and SPLIT PARTITION.
5. Users can perform the following DDL operations on the current binlog table. However, the incremental data and synchronization point information will be cleared.
ADD COLUMN, DROP COLUMN, SET TYPE, and TRUNCATE
6. The system waits for binlog consumption before further scaling. The default wait time is 1 hour. Timeouts or errors will cause the scaling process to fail.
7. The system waits for the consumption of binlog records before the **VACUUM FULL** operation is performed on a binlog table. The default wait time is 1 hour. Timeouts or errors will cause the **VACUUM FULL** process to fail. Additionally, even if VACUUM FULL is executed for a partitioned table, a level-7 lock is added to the primary table of the partition, which blocks the insertion, update, or deletion of the entire table.
8. Binlog tables are backed up as standard HStore tables. Post-restoration, you must restart data synchronization as incremental data and sync details are reset.
9. The Binlog timestamp function is supported. This function can be enabled by activating [enable_binlog_timestamp](#). Only the HStore and HStore Opt tables support this function. This constraint is supported only in 9.1.0.200 and later versions.

Binlog Formats and Principles

Table 17-6 binlog fields

| Field | Type | Description |
|--------------------------|--------|--|
| gs_binlog_sync_point | BIGINT | Binlog system field, which indicates the synchronization point. In common GTM mode, the value is unique and ordered. |
| gs_binlog_event_sequence | BIGINT | Binlog system field, which indicates the sequence of operations of the same transaction type. |

| Field | Type | Description |
|------------------------|-------------|--|
| gs_binlog_event_type | CHAR | Binlog system field, which indicates the operation type of the current record.
The options are as follows: <ul style="list-style-type: none">• I refers to INSERT, indicating that a new record is inserted into the current binlog.• d refers to DELETE, indicating that a record is deleted from the current binlog.• B refers to BEFORE_UPDATE, indicating that the current binlog is a record before the update.• U refers to AFTER_UPDATE, indicating that the current binlog is a record after the update. |
| gs_binlog_timestamp_us | BIGINT | System field of Binlog, indicating the timestamp when the current record is saved to the database.
This field is available only when the Binlog timestamp function is enabled. If the Binlog timestamp function is disabled, this field is left blank. Only 9.1.0.200 and later versions support this function. |
| user_column_1 | User column | User-defined data column |
| ... | ... | ... |
| user_column_n | User column | User-defined data column |

 NOTE

- For each UPDATE (or UPSERT-triggered update), two binlog records—BEFORE_UPDATE and AFTER_UPDATE—are created. BEFORE_UPDATE verifies the accuracy of data processed by third-party components like Flink.
- During UPDATE and DELETE operations, the DWS hybrid data warehouse generates BEFORE_UPDATE and DELETE binlogs without querying or populating all user columns, enhancing database import efficiency.
- Enabling binlog for an HStore table in the DWS hybrid data warehouse is the process of creation of a supplementary table. This table includes three system columns **gs_binlog_event_sync_point**, **gs_binlog_event_event_sequence**, and **gs_binlog_event_type**, and a **value** column that serializes all user columns.
- When the [enable_binlog_timestamp](#) parameter is enabled, binlog records are retained until the TTL expires, causing extra space overhead proportional to the data volume updated within the TTL. When **enable_binlog** is enabled, binlogs can be cleared asynchronously once consumed by downstream processes, significantly reducing space usage. Only 9.1.0.200 and later versions support this function.

Enabling Binlog

- Specify the table-level parameter **enable_binlog** when creating an HStore Opt table to enable binlogs.

```
CREATE TABLE hstore_binlog_source (  
  c1 INT PRIMARY KEY,  
  c2 INT,  
  c3 INT  
) WITH (  
  ORIENTATION = COLUMN,  
  enable_hstore_opt=true,  
  enable_binlog=on,  
  binlog_ttl = 86400  
);
```

NOTE

- Binlog recording begins only after a synchronization point is registered for the task, not during the initial data import. Once binlog synchronization in Flink is activated, it periodically acquires the synchronization point and incremental data, then registers the synchronization point.
 - The **binlog_ttl** parameter defaults to 86,400 seconds and is optional. If a registered synchronization point exceeds this TTL without undergoing incremental synchronization, it will be cleared. Subsequently, binlogs before the oldest synchronization point are asynchronously deleted to free up space.
- Space overhead: For a table with common binlog enabled, if incremental data can be consumed by downstream processes in a timely manner, the space can be cleared and reclaimed promptly.
- Run the **ALTER** command to enable the binlog function for an existing HStore table.

```
CREATE TABLE hstore_binlog_source (  
  c1 INT PRIMARY KEY,  
  c2 INT,  
  c3 INT  
) WITH (  
  ORIENTATION = COLUMN,  
  enable_hstore_opt=true  
);  
ALTER TABLE hstore_binlog_source SET (enable_binlog=on);
```

Querying Binlogs

- Use the system functions provided by DWS to query the binlog information of the target table on a specified DN and check whether the binlog is consumed by downstream processes.

```
-- Simulate Flink to call a system function to obtain the synchronization point. The parameters  
-- indicate the table name, slot name, whether the point is a checkpoint, and target DN (0 indicates all  
-- DNs).  
select * from pg_catalog.pgxc_get_binlog_sync_point('hstore_binlog_source', 'slot1', false, 0);  
select * from pg_catalog.pgxc_get_binlog_sync_point('hstore_binlog_source', 'slot1', true, 0);  
-- Incremental binlogs are generated after additions, deletions, and modifications.  
INSERT INTO hstore_binlog_source VALUES(100, 1, 1);  
delete hstore_binlog_source where c1 = 100;  
INSERT INTO hstore_binlog_source VALUES(200, 1, 1);  
update hstore_binlog_source set c2 =2 where c1 = 200;  
-- Simulate Flink to call a system function to query the binlog of a specified CSN range. The  
-- parameters indicate the table name, target DN (0 indicates all DNs), start CSN point, and end CSN  
-- point.  
select * from pgxc_get_binlog_changes('hstore_binlog_source', 0, 0 , 999999999);
```

```
postgres=# select * from pgxc_get_binlog_changes('hstore_binlog_source', 0, 0, 999999999);
gs_binlog_sync_point | gs_binlog_event_sequence | gs_binlog_event_type | gs_binlog_timestamp_us | c1 | c2 | c3
-----
10241 | 2 | I | | 100 | 1 | 1
10242 | 3 | d | | 100 | 1 | 1
10243 | 4 | I | | 100 | 1 | 1
10245 | 5 | B | | 100 | 1 | 1
10245 | 6 | U | | 100 | 100 | 1
(5 rows)
```

Two **INSERT** operations generate two records with **gs_binlog_event_type** as **I**. The **DELETE** operation generates a record whose type is **d**. The **UPDATE** operation generates a **B** record for **BeforeUpdate** and a **U** record for **AfterUpdate**, indicating the values before and after the update.

- Query the **pgxc_get_binlog_slots(rel_name text, node_id int)** function to obtain the synchronization information of all slots of the target table. The parameters indicate the target table name and target DN (**0** indicates all DNs are queried).

```
postgres=# select * from pgxc_get_binlog_slots('binlog_view_part_t1', 0);
node_name | slot_name | sync_point | sync_timestamp | oldest_xmin | is_checkpoint
-----
dn_1 | slot1 | 210059 | 2025-09-19 10:31:38.894357+08 | 10054 | f
(1 row)
```

- Call the system function **pgxc_consumed_binlog_records** to check whether the binlogs of the target table are consumed by all slots. The parameters indicate the target table name and target DN (**0** indicates all DNs are queried).

```
-- Simulate Flink to call the system function to register a synchronization point. The parameters
-- indicate the table name, slot name, registered point, whether the point is a checkpoint, and xmin
-- corresponding to the point (provided when the synchronization point is obtained).
select pgxc_register_binlog_sync_point('hstore_binlog_source', 'slot1', 0, 999999999, false, 100);
select pgxc_register_binlog_sync_point('hstore_binlog_source', 'slot1', 0, 999999999, true, 100);
-- Check whether all binlogs in the table are consumed. If 1 is returned, all binlogs have been
-- consumed by downstream slots.
select * from pgxc_consumed_binlog_records('hstore_binlog_source', 0);
```

```
postgres=# select * from pgxc_consumed_binlog_records('hstore_binlog_source', 0);
pgxc_consumed_binlog_records
-----
1
(1 row)
```

- Call the system function **pgxc_binlog_clear_target_slot(rel_name text, slot_name text)** to clear the specified slot on the specified table. The parameters indicate the target table name and target slot name, respectively.

Enabling the Binlog Timestamp Function

If you need to read binlogs generated after a specified time point, specify the table-level parameter **enable_binlog_timestamp** when creating an HStore table to enable the binlog timestamp function of the HStore table. Only 9.1.0.200 and later versions support this function.

```
CREATE TABLE hstore_binlog_source(
  c1 INT PRIMARY KEY,
  c2 INT,
  c3 INT
) WITH (
  ORIENTATION = COLUMN,
  enable_hstore_opt=true,
  enable_binlog_timestamp =on,
  binlog_ttl = 86400
);
```

 NOTE

- Binlog recording begins only after a synchronization point is registered for the task, not during the initial data import. Once the binlog timestamp is enabled, the system periodically acquires the synchronization point and incremental data, then registers the synchronization point.
- Binlog_ttl is an optional parameter. If not set, the default value is **86400** seconds (i.e., data is retained for one day by default). If the timestamp of the binlog record is greater than the current TTL, the binlog record will be deleted asynchronously.
- Space overhead: For a table with the binlog timestamp enabled, the binlog records recorded in the auxiliary table are retained until the TTL expires. This results in extra space overhead, which is proportional to the amount of data updated and imported into the database within the TTL.
- Query the binlog on the table where binlog timestamp is enabled.

```
postgres=# select * from pgxc_get_binlog_changes('hstore_binlog_source', 0, 0, 999999999);
gs_binlog_sync_point | gs_binlog_event_sequence | gs_binlog_event_type | gs_binlog_timestamp_us | c1 | c2 | c3
-----
```

| | | | | | | |
|-------|---|---|------------------|-----|---|---|
| 10516 | 2 | I | 1731570520900211 | 100 | 1 | 1 |
| 10517 | 3 | d | 1731570520904425 | 100 | 1 | 1 |
| 10518 | 2 | I | 1731570520909055 | 200 | 1 | 1 |
| 10519 | 3 | B | 1731570520914102 | 200 | 1 | 1 |
| 10519 | 4 | U | 1731570520914154 | 200 | 2 | 1 |

```
(5 rows)
```

Convert **gs_binlog_timestamp_us** from the BigInt type to a readable timestamp.

```
SELECT to_timestamp(1731569598408661/1000000);
```

```
postgres=# select to_timestamp(1731570520900211/1000000);
to_timestamp
-----
2024-11-14 15:48:40.900211+08
(1 row)
```

- Obtain the first binlog information of the target table after the specified time point on each DN (if the value is empty, no binlog exists after the time point).
SELECT * FROM pgxc_get_binlog_cursor_by_timestamp('hstore_binlog_source','2024-11-14 15:33:18.40866+08', 0);

```
postgres=# select * from pgxc_get_binlog_cursor_by_timestamp('hstore_binlog_source','2024-11-14 15:48:40.900211+08', 0);
node_name | node_id | latest_sync_point | binlog_sync_point | binlog_timestamp_us | binlog_xmin
-----
```

| | | | | | |
|------|-------------|-------|-------|------------------|-------|
| dn_2 | -1051926843 | 10532 | 10516 | 1731570520900211 | 10510 |
| dn_1 | -1300059100 | 10532 | 10518 | 1731570520909055 | 10510 |

```
(2 rows)
```

- Obtain the consumption progress of the table for which binlog timestamp is enabled.

The returned fields indicate the timestamp of the latest consumed binlog, the latest timestamp on the binlog, the CSN point of the latest consumed binlog, the latest CSN point on the binlog, and the number of unconsumed binlog records.

Query the consumption progress of each slot in the target table.

```
SELECT * FROM pgxc_get_binlog_consume_progress('hstore_binlog_source', 0);
```

```
postgres=# select * from pgxc_get_binlog_consume_progress('hstore_binlog_source', 0);
node_name | node_id | slot_name | checkpoint | latest_consumed_timestamp | latest_timestamp | latest_consumed_csn | latest_csn | unconsumed_binlog_count
-----
```

| | | | | | | | | |
|------|-------------|-------|---|------------------------|------------------------|-------|-------|---|
| dn_1 | -1300059100 | slot1 | f | 2024-11-14 15:48:40+08 | 2024-11-14 15:48:40+08 | 10519 | 10519 | 0 |
| dn_1 | -1300059100 | slot1 | f | 2024-11-14 15:48:40+08 | 2024-11-14 15:48:40+08 | 10519 | 10519 | 0 |
| dn_2 | -1051926843 | slot1 | f | 2024-11-14 15:48:40+08 | 2024-11-14 15:48:40+08 | 10517 | 10517 | 0 |
| dn_2 | -1051926843 | slot1 | t | 2024-11-14 15:48:40+08 | 2024-11-14 15:48:40+08 | 10517 | 10517 | 0 |

```
(4 rows)
```

Preventing DML from Generating Binlogs

Set the session-level parameter **enable_generate_binlog** to **off** to control the DML of the current session. When a table for which binlog is enabled is imported to the database, no binlog record is generated.

17.5.2 Real-Time Binlog Consumption by Flink

Precautions

- Currently, only 8.3.0.100 and later versions support HStore and HStore Opt tables to record binlogs.
- V3 HStore tables do not support binlogs, whereas only V3 HStore Opt tables offer binlog support. V3 is currently in trial commercial use and should be thoroughly evaluated before deployment.
- The Binlog function is only supported for HStore and HStore Opt tables in DWS. These tables must have primary keys and one of parameters **enable_binlog** and **enable_binlog_timestamp** must be set to **on**.
- The name of the consumed binlog table cannot contain special characters, such as periods (.) and double quotation marks (").
- If multiple tasks consume binlog data of a single table, ensure that **binlogSlotName** of each task is unique.
- For maximum consumption speed, match task concurrency with the number of DN in your DWS cluster.
- If you use the sink capability of "dws-connector-flink" in *Data Warehouse Service (DWS) Tool Guide* to write binlog data, pay attention to the following:
 - To ensure the data write sequence on DN, set **connectionSize** to **1**.
 - If the primary key is updated on the source end or Flink is required for aggregation calculation, set **ignoreUpdateBefore** to **false**. Otherwise, you are not advised to set **ignoreUpdateBefore** to **false** (the default value is **true**).

Real-Time Binlog Consumption by Flink

Use DWS Connector to consume binlogs in real time. For details, see "DWS-Connector" in *Data Warehouse Service (DWS) Tool Guide*.

If full data has been synchronized to the target end using other synchronization tools, and only incremental synchronization is required, you can call the following system function to update the synchronization points.

```
SELECT * FROM pg_catalog.pgxc_register_full_sync_point('table_name', 'slot_name');
```

Source Table DDL

The source autonomously assigns the appropriate Flink RowKind type (INSERT, DELETE, UPDATE_BEFORE, or UPDATE_AFTER) to each data row based on the operation type. This mechanism facilitates the synchronization of table data in a mirrored way, akin to the Change Data Capture (CDC) feature in MySQL and PostgreSQL databases.

```
CREATE TABLE test_binlog_source (  
  a int,  
  b int,  
  c int,  
  primary key(a) NOT ENFORCED  
) with (  
  'connector' = 'dws',  
  'url' = 'jdbc:gaussdb://ip:port/gaussdb',  
  'binlog' = 'true',
```

```
'tableName' = 'test_binlog_source',  
'binlogSlotName' = 'slot',  
'username'='xxx',  
'password'='xxx')
```

Binlog Parameters

The following table describes the parameters involved in binlog consumption.

Table 17-7 Parameters

| Parameter | Description | Data Type | Default Value |
|-----------------------------|--|-----------|---------------------------------|
| binlog | Specifies whether to read binlog information. | Boolean | false |
| binlogSlotName | Slot, which serves as an identifier. Multiple Flink tasks can simultaneously consume binlog data of the same table, so each task's binlogSlotName must be unique.

If binlog is specified but binlogSlotName is not specified, the table name of the Flink mapping table is used by default to ensure that incremental data is correctly read. | String | Name of the Flink mapping table |
| binlogBatchRead-Size | Rows of binlog data read in batches. | Integer | 5000 |
| fullSyncBinlogBatchReadSize | Rows of binlog data fully read. | Integer | 50000 |
| binlogReadTimeout | Timeout for incrementally consuming binlog data, in milliseconds. | Integer | 600000 |
| fullSyncBinlogRead-Timeout | Timeout for fully consuming binlog data, in milliseconds. | Long | 1800000 |
| binlogSleepTime | Sleep duration when no real-time binlog data is consumed, in milliseconds. The sleep duration with consecutive read failures is binlogSleepTime * failures, up to binlogMaxSleepTime . The value is reset after successful data read. | Long | 500 |
| binlogMaxSleepTime | Maximum sleep duration when no real-time binlog data is consumed, in milliseconds. | Long | 10000 |

| Parameter | Description | Data Type | Default Value |
|-------------------------|---|-----------|---------------|
| binlogMaxRetryTimes | Maximum number of retries after a binlog data consumption error. | Integer | 1 |
| binlogRetryInterval | Interval between retries after a binlog data consumption error, in milliseconds. Sleep duration during retry, which is calculated as binlogRetryInterval * (1~ binlogMaxRetryTimes) + Random(100). The unit is millisecond. | Long | 100 |
| binlogParallelNum | Number of threads for consuming binlog data. This parameter is valid only when task concurrency is less than the number of DNs in the DWS cluster. | Integer | 3 |
| connectionPoolSize | Number of connections in the JDBC connection pool. | Integer | 5 |
| needRedistribution | Determines compatibility with expansion redistribution. To ensure compatibility, upgrade the kernel to the corresponding version. If the kernel is an older version, set this parameter to false . If set to true , restart-strategy of Flink cannot be set to none . | Boolean | true |
| newSystemValue | Indicates whether to use the new system field when reading binlog data. (The kernel needs to be upgraded to the corresponding version. If the kernel is an older version, set this parameter to false .) | Boolean | true |
| checkNodeChangeInterval | Interval for detecting node changes. This parameter is valid only when needRedistribution is set to true . | Long | 10000 |

| Parameter | Description | Data Type | Default Value |
|--------------------------|--|-----------|---------------|
| connectionSocket-Timeout | Timeout interval for connection processing, in milliseconds. It can also be considered as the timeout interval for executing SQL statements on the client. The default value is 0 , which means that the timeout interval is not set. | Integer | 0 |
| binlogIgnoreUpdateBefore | Determines whether to filter out before_update records in binlogs and whether to return only primary key information for delete records. This parameter is supported only in 9.1.0.200 and later versions. | Boolean | false |
| binlogStartTime | Sets the time point from which binlogs are consumed can be set using the format yyyy-MM-dd hh:mm:ss .
enable_binlog_timestamp must be enabled for the table.
This parameter is supported only in 9.1.0.200 and later versions. | String | N/A |
| binlogSyncPointSize | Specifies the size of the synchronization point range for incrementally reading binlogs. This can control data flushing if the data volume is too large.
This parameter is supported only in 9.1.0.200 and later versions. | Integer | 5000 |

Data Synchronization Example

- DWS:

NOTE

When creating a binlog table, set **enable_hstore_binlog_table** to **true**. You can run the **show enable_hstore_binlog_table** command to query the binlog table.

-- Source table (generating binlogs)

```
CREATE TABLE test_binlog_source(a int, b int, c int, primary key(a)) with(orientation=column, enable_hstore_opt=on, enable_binlog=true);
```

-- Target table

```
CREATE TABLE test_binlog_sink(a int, b int, c int, primary key(a)) with(orientation=column, enable_hstore_opt=on);
```

- On Flink:

Run the following commands to perform complete data synchronization:

```
-- Create a mapping table for the source table.
CREATE TABLE test_binlog_source (
  a int,
  b int,
  c int,
  primary key(a) NOT ENFORCED
) with (
  'connector' = 'dws',
  'url' = 'jdbc:gaussdb://ip:port/gaussdb',
  'binlog' = 'true',
  'tableName' = 'test_binlog_source',
  'binlogSlotName' = 'slot',
  'username'='xxx',
  'password'='xxx');

-- Create a mapping table for the target table:
CREATE TABLE test_binlog_sink (
  a int,
  b int,
  c int,
  primary key(a) NOT ENFORCED
) with (
  'connector' = 'dws',
  'url' = 'jdbc:gaussdb://ip:port/gaussdb',
  'tableName' = 'test_binlog_sink',
  'ignoreUpdateBefore'='false',
  'connectionSize' = '1',
  'username'='xxx',
  'password'='xxx');

INSERT INTO test_binlog_sink select * from test_binlog_source;
```

Example of Using Java Programs

Create a source table and a target table.

```
-- source
create table binlog_test_source(a int, b int, c int, primary key(a)) with(orientation=column,
enable_hstore_opt=on, enable_binlog=true);
-- sink
create table binlog_test_sink(a int, b int, c int, primary key(a)) with(orientation=column,
enable_hstore_opt=on, enable_binlog=true);
```

Demo program:

```
public class BinlogDemo {

    //Name of the binlog table
    private static final String BINLOG_TABLE_NAME = "binlog_test_source";

    //Slot name of the binlog table
    private static final String BINLOG_SLOT_NAME = "binlog_test_slot";

    //Name of the table to be written
    private static final String SINK_TABLE_NAME = "binlog_test_sink";

    public static void main(String[] args) throws Exception {
        DwsConfig dwsConfig = buildDwsConfig();
        DwsClient dwsClient = new DwsClient(dwsConfig);

        TableSchema sourceTableSchema =
dwsClient.getTableSchema(TableName.valueOf(BINLOG_TABLE_NAME));
        TableSchema sinkTableSchema = dwsClient.getTableSchema(TableName.valueOf(SINK_TABLE_NAME));

        // Columns to be written
        List<String> sinkColumns = sinkTableSchema.getColumnNames();
```

```
// Thread pool
DwsConnectionPool dwsConnectionPool = new DwsConnectionPool(dwsConfig);
//Queue for storing data
BlockingQueue<BinlogRecord> queue = new LinkedBlockingQueue<>();
//Columns to be synchronized
List<String> sourceColumnNames = sourceTableSchema.getColumnNames();

BinlogReader binlogReader = new BinlogReader(dwsConfig, queue, sourceColumnNames,
dwsConnectionPool);

//Start the read task.
binlogReader.start();
binlogReader.getRecords();

while (binlogReader.isStart()) {
    try {
        while (!queue.isEmpty() && !binlogReader.hasException()) {
            // Read data.
            BinlogRecord record = queue.poll();
            if (Objects.isNull(record)) {
                continue;
            }
            BinlogRecordType type = BinlogRecordType.toBinlogRecordType(record.getType());
            List<Object> columnValues = record.getColumnValues();

            // Write data.
            if (BinlogRecordType.INSERT.equals(type) || BinlogRecordType.UPDATE_AFTER.equals(type)) {
                Operate upsert = dwsClient.write(sinkTableSchema);
                for (int i = 0; i < sinkColumns.size(); i++) {
                    upsert.setObject(i, columnValues.get(i), false);
                }
                upsert.commit();
            } else if (BinlogRecordType.DELETE.equals(type) ||
BinlogRecordType.UPDATE_BEFORE.equals(type)) {
                Operate delete = dwsClient.delete(sinkTableSchema);
                for (int i = 0; i < sinkColumns.size(); i++) {
                    String field = sinkColumns.get(i);
                    if (!sinkTableSchema.isPrimaryKey(field)) {
                        continue;
                    }
                    delete.setObject(i, columnValues.get(i), false);
                }
                delete.commit();
            }
        }
        binlogReader.checkException();
    } catch (Exception e) {
        throw new DwsClientException(ExceptionCode.GET_BINLOG_ERROR, "get binlog has error", e);
    }
}

private static DwsConfig buildDwsConfig() {
    //Initialize configuration information. (Only necessary parameters are listed. For more information
about the configuration, see the section.)
    TableConfig tableConfig = new TableConfig().withBinlog(true)
        .withNewSystemValue(true)
        .withNeedRedistribution(false)
        .withBinlogSlotName(BINLOG_SLOT_NAME);
    return DwsConfig.builder()
        .withUrl("Link information")
        .withUsername("Username")
        .withPassword("Password")
        .withBinlogTableName(BINLOG_TABLE_NAME)
        .withTableConfig(BINLOG_TABLE_NAME, tableConfig)
        .build();
}
```

18 DWS System Catalogs and Views

18.1 Overview of System Catalogs and System Views

System catalogs are used by DWS to store structure metadata. They are a core component the DWS database system and provide control information for the database system. These system catalogs contain cluster installation information and information about various queries and processes in DWS. You can collect information about the database by querying the system catalog.

System views provide ways to query system catalogs and internal database status. If some columns in one or more tables in a database are frequently searched for, an administrator can define a view for these columns, and then users can directly access these columns in the view without entering search criteria. A view is different from a basic table. It is only a virtual object rather than a physical one. A database only stores the definition of a view and does not store its data. The data is still stored in the original base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database. A view is triggered every time it is referenced.

In separation of duty, non-administrators have no permission to view system catalogs and views. In other scenarios, system catalogs and views are either visible only to administrators or visible to all users. Some of the following system catalogs and views have marked the need of administrator permissions. They are accessible only to administrators.

NOTICE

- Do not add, delete, or modify system catalogs or system views. Manual modification or damage to system catalogs or system views may cause system information inconsistency, system control exceptions, or even cluster unavailability.
 - System catalogs do not support toast and cannot be stored across pages. The size of a page is 8 KB, and the length of each field in the system catalog must be less than 8 KB.
-

18.2 System Catalogs

18.2.1 GS_BLOCKLIST_QUERY

GS_BLOCKLIST_QUERY records job blocklist and exception information. This table uses **unique_sql_id** as the unique index to collect statistics on job exception information and record blocklist information. You can associate **GS_BLOCKLIST_QUERY** with **GS_WLM_SESSION_INFO** to obtain the **query** column and execution information of a job.

NOTICE

- The schema of the **GS_BLOCKLIST_QUERY** system catalog is **dbms_om**.
- The **GS_BLOCKLIST_QUERY** system catalog can be queried only in the **postgres** database. If it is queried in other databases, an error is reported.
- The **GS_BLOCKLIST_QUERY** system catalog contains unique indexes, which are distributed on DNs in hash mode. The distributed column is **unique_sql_id**.
- Generally, constant values are ignored during Unique SQL ID calculation in DML statements. However, constant values cannot be ignored in DDL, DCL, and parameter setting statements. A **unique_sql_id** may correspond to one or more queries.

DWS also provides the **GS_BLOCKLIST_QUERY** view for querying job blocklist and exception information. This view can directly display the **query** column. This view depends on **GS_WLM_SESSION_INFO**. If the **GS_WLM_SESSION_INFO** table is large, the query may take a long time.

Table 18-1 GS_BLOCKLIST_QUERY columns

| Name | Type | Reference | Description |
|---------------|-----------|-----------|--|
| unique_sql_id | Bigint | N/A | Query ID generated based on the query parsing tree. |
| block_list | Boolean | N/A | Check whether a job is in the blocklist. |
| except_num | Integer | N/A | Query the number of job exceptions. |
| except_time | Timestamp | N/A | Query the time when the last job exception occurred. |

18.2.2 GS_BLOCKLIST_SQL

GS_BLOCKLIST_SQL records job blocklist and exception information. This table uses **sql_hash** as the unique index to collect statistics on job exception

information and record blocklist information. You can associate **GS_BLOCKLIST_SQL** with **GS_WLM_SESSION_INFO** to obtain the **query** column and execution information of a job.

DWS also provides the **GS_BLOCKLIST_SQL** view for querying job blocklist and exception information. This view can directly display the **query** column. This view depends on **GS_WLM_SESSION_INFO**. If the **GS_WLM_SESSION_INFO** table is large, the query may take a long time.

This system catalog is supported only by clusters of version 9.1.0.200 or later.

NOTICE

- The schema of the **GS_BLOCKLIST_SQL** system catalog is **dbms_om**.
- The **GS_BLOCKLIST_SQL** system catalog can be queried only in the **postgres** database. If it is queried in other databases, an error is reported.
- The **GS_BLOCKLIST_SQL** system catalog contains unique indexes, which are distributed on DN's in hash mode. The distributed column is **sql_hash**.
- Generally, constant values are ignored during **sql_hash** calculation in DML statements. However, constant values cannot be ignored in DDL, DCL, and parameter setting statements. A **sql_hash** may correspond to one or more queries.

Table 18-2 GS_BLOCKLIST_SQL columns

| Name | Type | Reference | Description |
|-------------|-----------|-----------|--|
| sql_hash | Text | N/A | sql_hash generated based on the query parsing tree. |
| block_list | Boolean | N/A | Check whether a job is in the blocklist. |
| except_num | Integer | N/A | Query the number of job exceptions. |
| except_time | Timestamp | N/A | Query the time when the last job exception occurred. |

18.2.3 GS_OBSSCANINFO

GS_OBSSCANINFO defines the OBS runtime information scanned in cluster acceleration scenarios. Each record corresponds to a piece of runtime information of a foreign table on OBS in a query.

Table 18-3 GS_OBSSCANINFO columns

| Name | Type | Description |
|--------------|--------------------------|--|
| query_id | Bigint | Specifies a query ID. |
| user_id | Text | Specifies a database user who performs queries. |
| table_name | Text | Specifies the name of a foreign table on OBS. |
| file_type | Text | Specifies the format of files storing the underlying data. |
| time_stamp | timestamp with time zone | Specifies the scanning start time. |
| actual_time | double | Specifies the scanning execution time in seconds. |
| file_scanned | Bigint | Specifies the number of files scanned. |
| data_size | double | Specifies the size of data scanned in bytes. |
| billing_info | Text | Specifies the reserved fields. |

18.2.4 GS_RESPOOL_RESOURCE_HISTORY

The **GS_RESPOOL_RESOURCE_HISTORY** table records the historical monitoring information about a resource pool on both CNs and DN.

Table 18-4 GS_RESPOOL_RESOURCE_HISTORY columns

| Name | Type | Description |
|-----------|-----------|--|
| Timestamp | Timestamp | Time when resource pool monitoring information is persistently stored |
| nodegroup | Name | Name of the logical cluster of the resource pool. The default value is installation . |
| rpname | Name | Resource pool name |
| cgroup | Name | Name of the Cgroup associated with the resource pool |
| ref_count | Int | Number of jobs referenced by the resource pool. The number is counted regardless of whether the jobs are controlled by the resource pool. This parameter is valid only on CNs. |
| fast_run | Int | Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs. |

| Name | Type | Description |
|-----------------|--------|--|
| fast_wait | Int | Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs. |
| fast_limit | Int | Limit on the number of concurrent jobs in the fast lane in a resource pool. This parameter is valid only on CNs. |
| slow_run | Int | Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs. |
| slow_wait | Int | Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs. |
| slow_limit | Int | Limit on the number of concurrent jobs in the slow lane in a resource pool. This parameter is valid only on CNs. |
| used_cpu | Double | <p>Average number of CPUs used by the resource pool in a 5s monitoring period. The value is accurate to two decimal places.</p> <ul style="list-style-type: none">On a DN, it indicates the number of CPUs used by the resource pool on the current DN.On a CN, it indicates the total CPU usage of resource pools on all DNs. |
| cpu_limit | Int | <p>It indicates the upper limit of available CPUs for resource pools. If the CPU share is limited, this parameter indicates the available CPUs for DWS. If the CPU limit is specified, this parameter indicates the available CPUs for associated Cgroups.</p> <ul style="list-style-type: none">On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN.On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs. |
| used_mem | Int | <p>Memory used by the resource pool, in MB.</p> <ul style="list-style-type: none">On a DN, it indicates the memory usage of the resource pool on the current DN.On a CN, it indicates the total memory usage of resource pools on all DNs. |
| estimate_memory | Int | Estimated memory used by the jobs running in the resource pools on the current CN. This parameter is valid only on CNs. |

| Name | Type | Description |
|--------------|--------|--|
| mem_limit | Int | Upper limit of available memory for the resource pool (unit: MB). <ul style="list-style-type: none">On a DN, it indicates the upper limit of available memory for the resource pool on the current DN.On a CN, it indicates the total upper limit of available memory for resource pools on all DNs. |
| read_kbytes | Bigint | Number of logical read bytes in the resource pool within a 5s monitoring period (unit: KB). <ul style="list-style-type: none">On a DN, it indicates the number of logical read bytes in the resource pool on the current DN.On a CN, it indicates the total logical read bytes of resource pools on all DNs. |
| write_kbytes | Bigint | Number of logical write bytes in the resource pool within a 5s monitoring period (unit: KB). <ul style="list-style-type: none">On a DN, it indicates the number of logical write bytes in the resource pool on the current DN.On a CN, it indicates the total logical write bytes of resource pools on all DNs. |
| read_counts | Bigint | Number of logical reads in the resource pool within a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the number of logical reads in the resource pool on the current DN.On a CN, it indicates the total number of logical reads in resource pools on all DNs. |
| write_counts | Bigint | Number of logical writes in the resource pool within a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the number of logical writes in the resource pool on the current DN.On a CN, it indicates the total number of logical writes in resource pools on all DNs. |
| read_speed | Double | Average rate of logical reads of the resource pool in a 5s monitoring period, in KB/s. <ul style="list-style-type: none">On a DN, it indicates the logical read rate of the resource pool on the current DN.On a CN, it indicates the overall logical read rate of resource pools on all DNs. |

| Name | Type | Description |
|-------------|--------|--|
| write_speed | Double | Average rate of logical writes of resource pools in a 5s monitoring period, in KB/s. <ul style="list-style-type: none">On a DN, it indicates the logical write rate of the resource pool on the current DN.On a CN, it indicates the overall logical write rate of resource pools on all DNs. |
| send_speed | Double | Average network sending rate of the resource pool in a 5-second monitoring period, in KB/s. <ul style="list-style-type: none">On a DN, it indicates the network sending rate of the resource pool on the current DN.On a CN, it indicates that the cumulative sum of the network sending rates of the resource pool on all DNs. |
| recv_speed | Double | Average network receiving rate of the resource pool in a 5-second monitoring period, in KB/s. <ul style="list-style-type: none">On a DN, it indicates the network receiving rate of the resource pool on the current DN.On a CN, it indicates that the cumulative sum of the network receiving rates of the resource pool on all DNs. |

18.2.5 GS_WLM_INSTANCE_HISTORY

The **GS_WLM_INSTANCE_HISTORY** system catalog stores information about resource usage related to CN or DN instances. Each record in the system table indicates the resource usage of an instance at a specific time point, including the memory, number of CPU cores, disk I/O, physical I/O of the process, and logical I/O of the process.

Table 18-5 GS_WLM_INSTANCE_HISTORY column

| Column | Type | Description |
|--------------|--------------------------|---|
| instancename | Text | Instance name |
| Timestamp | Timestamp with time zone | Timestamp |
| used_cpu | Int | CPU usage of an instance |
| free_mem | Int | Unused memory of an instance (unit: MB) |
| used_mem | Int | Used memory of an instance (unit: MB) |

| Column | Type | Description |
|---------------|--------|---|
| io_wait | Real | Specifies the io_wait value (average value within 10 seconds) of the disk used by an instance. |
| io_util | Real | Specifies the io_util value (average value within 10 seconds) of the disk used by an instance. |
| disk_read | Real | Specifies the disk read rate (average value within 10 seconds) of an instance (unit: KB/s). |
| disk_write | Real | The disk write rate (average value within 10 seconds) of an instance (unit: KB/s). |
| process_read | Bigint | Specifies the read rate (excluding the number of bytes read from the disk pagecache) of the corresponding instance process that reads data from a disk. (Unit: KB/s) |
| process_write | Bigint | Specifies the write rate (excluding the number of bytes written to the disk pagecache) of the corresponding instance process that writes data to a disk within 10 seconds. (Unit: KB/s) |
| logical_read | Bigint | CN instance: N/A
DN instance: Specifies the logical read byte rate of the instance in the statistical interval (10 seconds). (Unit: KB/s) |
| logical_write | Bigint | CN instance: N/A
DN instance: Specifies the logical write byte rate of the instance within the statistical interval (10 seconds). (Unit: KB/s) |
| read_counts | Bigint | CN instance: N/A
DN instance: Specifies the total number of logical read operations of the instance in the statistical interval (10 seconds). |
| write_counts | Bigint | CN instance: N/A
DN instance: Specifies the total number of logical write operations of the instance in the statistical interval (10 seconds). |

18.2.6 GS_WLM_OPERATOR_INFO

GS_WLM_OPERATOR_INFO records operators of completed jobs. The data is dumped from the kernel to a system catalog. If the GUC parameter **enable_resource_record** is set to **on**, the system periodically imports records in **GS_WLM_OPERATOR_HISTORY** to this system catalog. You are not advised to enable this function because it occupies storage space and affects performance.

You are advised to disable it after performance locating and monitoring tasks are complete.

NOTICE

- The schema of the **GS_WLM_OPERATOR_INFO** system table is **dbms_om**.
- The **GS_WLM_OPERATOR_INFO** system catalog can be queried only in the **postgres** database. If it is queried in other databases, an error is reported.

Table 18-6 GS_WLM_OPERATOR_INFO columns

| Column | Type | Description |
|---------------------|--------------------------|--|
| nodename | Text | Name of the CN where the statement is executed |
| queryid | Bigint | Internal query_id used for statement execution |
| pid | Bigint | Backend thread ID |
| plan_node_id | Integer | plan_node_id of the execution plan of a query |
| plan_node_name | Text | Name of the operator corresponding to plan_node_id |
| start_time | Timestamp with time zone | Time when an operator starts to process the first data record |
| duration | Bigint | Total execution time of an operator. The unit is ms. |
| query_dop | Integer | Degree of parallelism (DOP) of the current operator |
| estimated_rows | Bigint | Number of rows estimated by the optimizer |
| tuple_processed | Bigint | Number of elements returned by the current operator |
| min_peak_memory | Integer | Minimum peak memory used by the current operator on all DNs. The unit is MB. |
| max_peak_memory | Integer | Maximum peak memory used by the current operator on all DNs. The unit is MB. |
| average_peak_memory | Integer | Average peak memory used by the current operator on all DNs. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of the current operator among DNs |

| Column | Type | Description |
|--------------------|---------|--|
| min_spill_size | Integer | Minimum spilled data among all DN's when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DN's when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DN's when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_cpu_time | Bigint | Minimum execution time of the operator on all DN's. The unit is ms. |
| max_cpu_time | Bigint | Maximum execution time of the operator on all DN's. The unit is ms. |
| total_cpu_time | Bigint | Total execution time of the operator on all DN's. The unit is ms. |
| cpu_skew_percent | Integer | Skew of the execution time among DN's. |
| warning | Text | Warning. The following warnings are displayed:
1. Sort/SetOp/HashAgg/HashJoin spill
2. The max spill size exceeds the alarm size 256MB
3. The max broadcast size exceeds the alarm size 100MB
4. Early spill
5. Spill times is greater than 3
6. Spill on memory adaptive
7. Hash table conflict
8. NestLoop in hashjoin |

18.2.7 GS_WLM_SESSION_INFO

GS_WLM_SESSION_INFO records load management information about a completed job executed on the current CN. The data is dumped from the kernel to a system catalog. If the GUC parameter [enable_resource_record](#) is set to **on**, the system periodically imports records in [GS_WLM_SESSION_HISTORY](#) to this system catalog. You are not advised to enable this function because it occupies storage space and affects performance. You are advised to disable it after performance locating and monitoring tasks are complete.

NOTICE

- The schema of the **GS_WLM_SESSION_INFO** system table is **dbms_om**.
- The **GS_WLM_SESSION_INFO** system catalog can be queried only in the **postgres** database. If it is queried in other databases, an error is reported.

Table 18-162 lists the columns in the **GS_WLM_SESSION_INFO** system catalog.

Table 18-7 GS_WLM_SESSION_HISTORY columns

| Column | Type | Description |
|------------------|--------------------------|--|
| datid | OID | OID of the database this backend is connected to |
| dbname | Text | Name of the database the backend is connected to |
| schemaname | Text | Schema name |
| nodename | Text | Name of the CN where the statement is run |
| username | Text | User name used for connecting to the backend |
| application_name | Text | Name of the application that is connected to the backend |
| client_addr | inet | IP address of the client connected to this backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used |
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |
| block_time | Bigint | Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is ms. |
| start_time | Timestamp with time zone | Time when the statement starts to be run |

| Column | Type | Description |
|---------------------|--------------------------|--|
| finish_time | Timestamp with time zone | Time when the statement execution ends |
| duration | Bigint | Execution time of a statement. The unit is ms. |
| estimate_total_time | Bigint | Estimated execution time of a statement. The unit is ms. |
| status | Text | Final statement execution status. Its value can be finished (normal) or aborted (abnormal). The statement status here is the execution status of the database server. If the statement is successfully executed on the database server but an error is reported in the result set, the statement status is finished . |
| abort_info | Text | Exception information displayed if the final statement execution status is aborted . |
| resource_pool | Text | Resource pool used by the user |
| control_group | Text | Cgroup used by the statement |
| estimate_memory | Integer | Estimated memory used by a statement on a single instance. The unit is MB. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DN. The unit is MB. |
| average_peak_memory | Integer | Average memory usage during statement execution. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of a statement among DN. |
| spill_info | Text | Statement spill information on all DN.
None indicates that the statement has not been spilled to disks on any DN.
All : The statement has been spilled to disks on all DN.
[a,b] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DN. |
| min_spill_size | Integer | Minimum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0 . |
| max_spill_size | Integer | Maximum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0 . |

| Column | Type | Description |
|---------------------|---------|---|
| average_spill_size | Integer | Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_dn_time | Bigint | Minimum execution time of a statement across all DNs. The unit is ms. |
| max_dn_time | Bigint | Maximum execution time of a statement across all DNs. The unit is ms. |
| average_dn_time | Bigint | Average execution time of a statement across all DNs. The unit is ms. |
| dntime_skew_percent | Integer | Execution time skew of a statement among DNs. |
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DNs. The unit is ms. |
| cpu_skew_percent | Integer | CPU time skew of a statement among DNs. |
| min_peak_iops | Integer | Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| max_peak_iops | Integer | Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| average_peak_iops | Integer | Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| iops_skew_percent | Integer | I/O skew across DNs. |

| Column | Type | Description |
|----------------|--------|---|
| warning | Text | Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed:
<ol style="list-style-type: none">1. The max spill size exceeds the alarm size 256MB2. The max broadcast size exceeds the alarm size 100MB3. Early spill4. Spill times is greater than 35. Spill on memory adaptive6. Hash table conflict7. NestLoop in hashjoin |
| queryid | Bigint | Internal query ID used for statement execution |
| query | Text | Statement to be executed. A maximum of 64 KB of strings can be retained. |
| query_plan | Text | Execution plan of a statement.
Specification restrictions:
<ol style="list-style-type: none">1. Execution plans are displayed only for DML statements.2. In 8.2.1.100 and later versions, the number of data binding times is added to the execution plans of Parse Bind Execute (PBE) statements to facilitate statement analysis. The number of data binding times is displayed in the format of PBE bind times: Times. |
| node_group | Text | Logical cluster of the user running the statement |
| pid | Bigint | PID of the backend thread of the statement |
| lane | Text | Fast/Slow lane where the statement is executed |
| unique_sql_id | Bigint | ID of the normalized unique SQL. |
| session_id | Text | Unique identifier of a session in the database system. Its format is session_start_time.tid.node_name . |
| min_read_bytes | Bigint | Minimum I/O read bytes of a statement across all DN. The unit is byte. |
| max_read_bytes | Bigint | Maximum I/O read bytes of a statement across all DN. The unit is byte. |

| Column | Type | Description |
|---------------------|--------|---|
| average_read_bytes | Bigint | Average I/O read bytes of a statement across all DNs. |
| min_write_bytes | Bigint | Minimum I/O write bytes of a statement across all DNs. |
| max_write_bytes | Bigint | Maximum I/O write bytes of a statement across all DNs. |
| average_write_bytes | Bigint | Average I/O write bytes of a statement across all DNs. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DNs. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DNs. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| stmt_type | Text | Query type corresponding to the statement. |
| except_info | Text | Information about the exception rule triggered by the statement. |
| unique_plan_id | Bigint | ID of the normalized unique plan. |
| sql_hash | Text | Normalized SQL hash. |
| plan_hash | Text | Normalized plan hash. |
| use_plan_baseline | Text | Indicates whether the bound plan is used for executing the current statement. If it is used, the name of the plan_baseline column in pg_plan_baseline is displayed. |
| outline_name | Text | Name of the outline used for the statement plan. |

| Column | Type | Description |
|-----------------------------|--------------|---|
| loader_status | Text | The JSON string for storing import and export service information is as follows.
<ol style="list-style-type: none">1. address: indicates the IP address of the peer cluster. The port number is displayed for the source cluster.2. direction: indicates the import and export service type. The value can be gds to file, gds from file, gds to pipe, gds from pipe, copy from or copy to.3. min/max/total_lines/bytes: indicates the minimum value, maximum value, total lines, and bytes of the import and export statements on all DNs. |
| parse_time | Bigint | Total parsing time before the statement is queued (including lexical and syntax parsing, optimization rewriting, and plan generation time), in milliseconds. This column is available only in clusters of version 8.3.0.100 or later. |
| disk_cache_hit_ratio | numeric(5,2) | Disk cache hit rate. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_read_size | Bigint | Total size of data read from disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_write_size | Bigint | Total size of data written to disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_size | Bigint | Total size of data read remotely from OBS due to disk cache read failure, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_time | Bigint | Total number of times data is read remotely from OBS due to disk cache read failure. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| vfs_scan_bytes | Bigint | Total number of bytes scanned by the OBS virtual file system in response to upper-layer requests, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|--|
| vfs_remote_read_bytes | Bigint | Total number of bytes actually read from OBS by the OBS virtual file system, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| preload_submit_time | Bigint | Total time for submitting I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_time | Bigint | Total time for waiting for I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_count | Bigint | Total number of times that the prefetching process waits for I/O requests. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| disk_cache_loaded_time | Bigint | Total time for reading from disk cache, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_conflict_count | Bigint | Number of times a block in the disk cache produces a hash conflict. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_error_count | Bigint | Number of disk cache read failures. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|---|
| disk_cache_error_code | Bigint | <p>Error code for disk cache read failures. Multiple error codes may be generated. If the disk cache fails to be read, OBS remote read is initiated and cache blocks are rewritten. The error code types are as follows: This column only applies to OBS 3.0 tables and foreign tables.</p> <ul style="list-style-type: none">• 1: A hash conflict occurs in the disk cache block.• 2: The generation time of the disk cache block is later than that of the OldestXmin transaction.• 4: Invoking the pread system when reading cache files from the disk cache failed.• 8: The data version of the disk cache block does not match.• 16: The version of the data written to the write cache does not match the latest version.• 32: Opening the cache file corresponding to the cache block failed.• 64: The size of the data read from the disk cache does not match.• 128: The CSN recorded in the disk cache block does not match. |
| obs_io_req_avg_rtt | Bigint | Average Round Trip Time (RTT) for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_avg_latency | Bigint | Average delay for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_1s | Bigint | Number of OBS I/O requests with a latency exceeding 1 second. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_10s | Bigint | Number of OBS I/O requests with a latency exceeding 10 seconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_count | Bigint | Total number of OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|-----------------------------|--------|---|
| obs_io_req_retry_count | Bigint | Total number of retries for OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_rate_limit_count | Bigint | Total number of times OBS I/O requests are flow-controlled. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

18.2.8 GS_WLM_USER_RESOURCE_HISTORY

The **GS_WLM_USER_RESOURCE_HISTORY** system catalog stores information about resources used by users. The data of this table is stored on both CNs and DN. Each record in the system table indicates the resource usage of a user at a time point, including the memory, number of CPU cores, storage space, temporary space, operator spill space, logical I/O traffic, number of logical I/O times, and logical I/O rate. The memory, CPU, and I/O monitoring items record only the resource usage of complex jobs.

Data in the **GS_WLM_USER_RESOURCE_HISTORY** system table comes from the [PG_TOTAL_USER_RESOURCE_INFO](#) view.

Table 18-8 GS_WLM_USER_RESOURCE_HISTORY column

| Column | Type | Description |
|--------------|--------------------------|--|
| username | Text | Username |
| Timestamp | Timestamp with time zone | Timestamp |
| used_memory | Int | Memory size used by a user, in MB. <ul style="list-style-type: none">DN: The memory used by users on the current DN is displayed.CN: The total memory usage of users on all DNs is displayed. |
| total_memory | Int | Memory used by the resource pool, in MB. 0 indicates that the available memory is not limited and depends on the maximum memory available in the database (max_dynamic_memory). A calculation formula is as follows:
$\text{total_memory} = \text{max_dynamic_memory} * \text{parent_percent} * \text{user_percent}$
CN: The sum of maximum available memory on all DNs is displayed. |

| Column | Type | Description |
|-------------------|--------|--|
| used_cpu | Real | Number of CPU cores in use |
| total_cpu | Int | Total number of CPU cores of the Cgroup associated with a user on the node |
| used_space | Bigint | Used storage space (unit: KB) |
| total_space | Bigint | Available storage space (unit: KB). -1 indicates that the storage space is not limited. |
| used_temp_space | Bigint | Used temporary storage space (unit: KB) |
| total_temp_space | Bigint | Available temporary storage space (unit: KB). -1 indicates that the maximum temporary storage space is not limited. |
| used_spill_space | Bigint | Space occupied by operators spilled to disk (unit: KB) |
| total_spill_space | Bigint | Available storage space for operator spill to disk (unit: KB). The value -1 indicates that the space is not limited. |
| read_kbytes | Bigint | Byte traffic of read operations in a monitoring period (unit: KB) |
| write_kbytes | Bigint | Byte traffic of write operations in a monitoring period (unit: KB) |
| read_counts | Bigint | Number of read operations in a monitoring period. |
| write_counts | Bigint | Number of write operations in a monitoring period. |
| read_speed | Real | Byte rate of read operations in a monitoring period (unit: KB) |
| write_speed | Real | Byte rate of write operations in a monitoring period (unit: KB) |
| send_speed | Double | Network sending rate in a monitoring period, in KB/s. |
| recv_speed | Double | Network receiving rate in a monitoring period, in KB/s. |

18.2.9 PG_AGGREGATE

pg_aggregate records information about aggregation functions. Each entry in **pg_aggregate** is an extension of an entry in **pg_proc**. The **pg_proc** entry carries

the aggregate's name, input and output data types, and other information that is similar to ordinary functions.

Table 18-9 PG_AGGREGATE columns

| Column | Type | Reference | Description |
|----------------|---------|---------------------------------|--|
| aggfnoid | regproc | PG_PROC.oid | PG_PROC OID of the aggregate function. |
| aggtransfn | regproc | PG_PROC.oid | Transition function. |
| aggcollectfn | regproc | PG_PROC.oid | Aggregate function. |
| aggfinalfn | regproc | PG_PROC.oid | Final function (zero if none). |
| aggsortop | OID | PG_OPERATOR.oid | Associated sort operator (zero if none). |
| aggtranstype | OID | PG_TYPE.oid | Data type of the aggregate function's internal transition (state) data. |
| agginitval | Text | - | Initial value of the transition state. This is a text column containing the initial value in its external string representation. If this column is null, the transition state value starts out null. |
| agginitcollect | Text | - | Initial value of the collection state. This is a text column containing the initial value in its external string representation. If this column is null, the collection state value starts out null. |

18.2.10 PG_AM

PG_AM records information about index access methods. There is one row for each index access method supported by the system.

Table 18-10 PG_AM columns

| Column | Type | Reference | Description |
|--------|------|-----------|---|
| OID | OID | - | Row identifier (hidden attribute; displayed only when explicitly selected). |
| amname | Name | - | Name of the access method. |

| Column | Type | Reference | Description |
|----------------|----------|-----------------------------|--|
| amstrategies | Smallint | - | Number of operator strategies for this access method, or zero if access method does not have a fixed set of operator strategies. |
| amsupport | Smallint | - | Number of support routines for this access method. |
| amcanorder | boolean | - | Whether the access method supports ordered scans sorted by the indexed column's value. |
| amcanorderbyop | boolean | - | Whether the access method supports ordered scans sorted by the result of an operator on the indexed column. |
| amcanbackward | boolean | - | Whether the access method supports backward scanning. |
| amcanunique | boolean | - | Whether the access method supports unique indexes. |
| amcanmulticol | boolean | - | Whether the access method supports multi-column indexes. |
| amoptionalkey | boolean | - | Whether the access method supports a scan without any constraint for the first index column. |
| amsearcharray | boolean | - | Whether the access method supports ScalarArrayOpExpr searches. |
| amsearchnulls | boolean | - | Whether the access method supports IS NULL/NOT NULL searches. |
| amstorage | boolean | - | Whether an index storage data type can differ from a column data type. |
| amclusterable | boolean | - | Whether an index of this type can be clustered on. |
| ampredlocks | boolean | - | Whether an index of this type manages fine-grained predicate locks. |
| amkeytype | OID | PG_TYPE.oid | Type of data stored in index, or zero if not a fixed type. |
| aminsert | regproc | PG_PROC.oid | "Insert this tuple" function. |

| Column | Type | Reference | Description |
|------------------|---------|-----------------------------|--|
| ambeginscan | regproc | PG_PROC.oid | "Prepare for index scan" function. |
| amgettupple | regproc | PG_PROC.oid | "Next valid tuple" function, or zero if none. |
| amgetbitmap | regproc | PG_PROC.oid | "Fetch all valid tuples" function, or zero if none. |
| amrescan | regproc | PG_PROC.oid | "(Re)start index scan" function. |
| amendscan | regproc | PG_PROC.oid | "Clean up after index scan" function. |
| ammarkpos | regproc | PG_PROC.oid | "Mark current scan position" function. |
| amrestrpos | regproc | PG_PROC.oid | "Restore marked scan position" function. |
| ammerge | regproc | PG_PROC.oid | "Merge multiple indexes" function. |
| ambuild | regproc | PG_PROC.oid | "Build new index" function. |
| ambuildempty | regproc | PG_PROC.oid | "Build empty index" function. |
| ambulkdelete | regproc | PG_PROC.oid | Bulk-delete function. |
| amvacuumclean up | regproc | PG_PROC.oid | Post- VACUUM cleanup function. |
| amcanreturn | regproc | PG_PROC.oid | Function to check whether index supports index-only scans, or zero if none |
| amcostestimate | regproc | PG_PROC.oid | Function to estimate cost of an index scan. |
| amoptions | regproc | PG_PROC.oid | Function to parse and validate reloptions for an index. |

18.2.11 PG_AMOP

PG_AMOP records information about operators associated with access method operator families. There is one row for each operator that is a member of an operator family. A family member can be either a search operator or an ordering operator. An operator can appear in more than one family, but cannot appear in more than one search position nor more than one ordering position within a family.

Table 18-11 PG_AMOP columns

| Name | Type | Reference | Description |
|----------------|----------|---------------------------------|---|
| OID | OID | - | Row identifier (hidden attribute; must be explicitly selected) |
| amopfamily | OID | PG_OPFAMILY.oid | Operator family this entry is for |
| amoplefttype | OID | PG_TYPE.oid | Left-hand input data type of operator |
| amoprightright | OID | PG_TYPE.oid | Right-hand input data type of operator |
| amopstrategy | Smallint | - | Number of operator strategies |
| amoppurpose | Char | - | Operator purpose, either s for search or o for ordering |
| amopopr | OID | PG_OPERATOR.oid | OID of the operator |
| amopmethod | OID | PG_AM.oid | Index access method the operator family is for |
| amopsortfamily | OID | PG_OPFAMILY.oid | If it is a sort operator, the item is sorted according to the B-Tree operator family. If it is a search operator, the value is 0. |

A "search" operator entry indicates that an index of this operator family can be searched to find all rows satisfying **WHERE indexed_column operator constant**. Obviously, such an operator must return a Boolean value, and its left-hand input type must match the index's column data type.

An "ordering" operator entry indicates that an index of this operator family can be scanned to return rows in the order represented by **ORDER BY indexed_column operator constant**. Such an operator could return any sortable data type, though again its left-hand input type must match the index's column data type. The exact semantics of **ORDER BY** are specified by the **amopsortfamily** column, which must reference a B-tree operator family for the operator's result type.

18.2.12 PG_AMPROC

PG_AMPROC records information about the support procedures associated with the access method operator families. There is one row for each support procedure belonging to an operator family.

Table 18-12 PG_AMPROC columns

| Name | Type | Reference | Description |
|-----------------|----------|---------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| amprocfamily | OID | PG_OPFAMILY.oid | Operator family this entry is for |
| amproclefttype | OID | PG_TYPE.oid | Left-hand input data type of associated operator |
| amprocrighttype | OID | PG_TYPE.oid | Right-hand input data type of associated operator |
| amprocnum | Smallint | N/A | Support procedure number |
| amproc | regproc | PG_PROC.oid | OID of the procedure |

The usual interpretation of the **amproclefttype** and **amprocrighttype** columns is that they identify the left and right input types of the operator(s) that a particular support procedure supports. For some access methods these match the input data type(s) of the support procedure itself, for others not. There is a notion of "default" support procedures for an index, which are those with **amproclefttype** and **amprocrighttype** both equal to the index opclass's **opcintype**.

18.2.13 PG_ATTRDEF

PG_ATTRDEF stores default values of columns.

Table 18-13 PG_ATTRDEF columns

| Column | Type | Description |
|-----------------|--------------|--|
| adrelid | OID | Table to which the column belongs |
| adnum | Smallint | Column No. |
| adbin | pg_node_tree | Internal representation of the column's default value |
| adsrc | Text | Internal representation of the human-readable default value |
| adbin_on_update | pg_node_tree | Internal representation of the value of on_update_expr |
| adsrc_on_update | Text | Internal representation of the human-readable value of on_update_expr |

18.2.14 PG_ATTRIBUTE

PG_ATTRIBUTE records information about table columns.

Table 18-14 PG_ATTRIBUTE columns

| Column | Type | Description |
|---------------|----------|---|
| attrelid | OID | Table to which the column belongs |
| attname | Name | Column name |
| atttypid | OID | Column type |
| attstattarget | Integer | Controls the level of details of statistics collected for this column by ANALYZE . <ul style="list-style-type: none">• A zero value indicates that no statistics should be collected.• A negative value says to use the system default statistics target.• The exact meaning of positive values is data type-dependent. For scalar data types, attstattarget is both the target number of "most common values" to collect, and the target number of histogram bins to create. |
| attlen | Smallint | Copy of pg_type.typlen of the column's type |
| attnum | Smallint | Number of a column. |
| attndims | Integer | Number of dimensions if the column is an array; otherwise, the value is 0. |
| attcacheoff | Integer | This column is always -1 on disk. When it is loaded into a row descriptor in the memory, it may be updated to cache the offset of the columns in the row. |
| atttypmod | Integer | Type-specific data supplied at table creation time (for example, the maximum length of a varchar column). This column is used as the third parameter when passing to type-specific input functions and length coercion functions. The value will generally be -1 for types that do not need ATTYPMOD . |
| attbyval | boolean | Copy of pg_type.typbyval of the column's type |
| attstorage | Char | Copy of pg_type.typstorage of this column's type |
| attalign | Char | Copy of pg_type.typalign of the column's type |

| Column | Type | Description |
|---------------|-----------|--|
| attnotnull | boolean | A not-null constraint. It is possible to change this column to enable or disable the constraint. |
| atthasdef | boolean | Indicates that this column has a default value, in which case there will be a corresponding entry in the pg_attrdef table that actually defines the value. |
| attisdropped | boolean | Whether the column has been dropped and is no longer valid. A dropped column is still physically present in the table but is ignored by the analyzer, so it cannot be accessed through SQL. |
| attislocal | boolean | Whether the column is defined locally in the relation. Note that a column can be locally defined and inherited simultaneously. |
| attcmprmode | tinyint | Compressed modes for a specific column. The compressed mode includes: <ul style="list-style-type: none">• ATT_CMPR_NOCOMPRESS• ATT_CMPR_DELTA• ATT_CMPR_DICTIONARY• ATT_CMPR_PREFIX• ATT_CMPR_NUMSTR |
| attinhcount | Integer | Number of direct ancestors this column has. A column with an ancestor cannot be dropped nor renamed. |
| attcollation | OID | Defined collation of a column |
| attacl | aclitem[] | Permissions for column-level access |
| attoptions | text[] | Property-level options |
| attfdwoptions | text[] | Property-level external data options |
| attinitdefval | bytea | attinitdefval stores the default value expression. ADD COLUMN in a row-store table must use this column. |

| Column | Type | Description |
|-----------|---------|--|
| attkvtype | tinyint | kv_type attribute of a column. Values: <ul style="list-style-type: none">• 0 indicates the default value, which is used for non-time series tables.• 1 indicates TSTAG, a dimension attribute, which is used only for time series tables.• 2 indicates TSFIELD, a metric attribute, which is used only for time series tables.• 3 indicates TSTIME, a time attribute, which is used only for time series tables. |

Example

Query the field names and field IDs of a specified table. Replace **t1** and **public** with the actual table name and schema name, respectively.

```
SELECT attname,attnum FROM pg_attribute WHERE attrelid=(SELECT pg_class.oid FROM pg_class JOIN pg_namespace ON relnamespace=pg_namespace.oid WHERE relname='t1' and nsname='public') and attnum>0;
```

| attname | attnum |
|------------------|--------|
| product_id | 1 |
| product_name | 2 |
| product_quantity | 3 |

(3 rows)

18.2.15 PG_AUTHID

PG_AUTHID records information about the database authentication identifiers (roles). The concept of users is contained in that of roles. A user is actually a role whose rolcanlogin has been set. Any role, whether the rolcanlogin is set or not, can use other roles as members.

For a cluster, only one **pg_authid** exists which is not available for every database. It is accessible only to users with system administrator rights.

Table 18-15 PG_AUTHID columns

| Column | Type | Description |
|---------------|---------|--|
| OID | OID | Row identifier (hidden attribute; must be explicitly selected) |
| rolname | Name | Role name |
| rolsuper | boolean | Whether the role is the initial system administrator with the highest permission |
| rolinherit | boolean | Whether the role automatically inherits permissions of roles it is a member of |
| rolcreaterole | boolean | Whether the role can create more roles |

| Column | Type | Description |
|----------------|--------------------------|--|
| rolcreatedb | boolean | Whether the role can create databases |
| rolcatupdate | boolean | Whether the role can directly update system catalogs. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users. |
| rolcanlogin | boolean | Whether a role can log in, that is, whether a role can be given as the initial session authorization identifier. |
| rolreplication | boolean | Indicates that the role is a replicated one (an adaptation syntax and no actual meaning). |
| rolauditadmin | boolean | Indicates that the role is an audit user. |
| rolsystemadmin | boolean | Indicates that the role is an administrator. |
| rolconndef | Integer | Limits the maximum number of concurrent connections of a user on a CN. -1 means no limit. |
| rolpassword | Text | Password (possibly encrypted); NULL if no password. |
| rolvalidbegin | Timestamp with time zone | Account validity start time; NULL if no start time |
| rolvaliduntil | Timestamp with time zone | Password expiry time; NULL if no expiration |
| rolrespool | Name | Resource pool that a user can use |
| roluseft | boolean | Whether the role can perform operations on foreign tables |
| rolparentid | OID | OID of a group user to which the user belongs |
| roltabspace | Text | Storage space of the user permanent table |
| rolkind | Char | Special type of user, including private users, logical cluster administrators, and common users. |
| rolnodegroup | OID | OID of a node group associated with a user. The node group must be a logical cluster. |
| roltemp space | Text | Storage space of the user temporary table |
| rolspill space | Text | Operator disk spill space of the user |
| rolexcpdata | Text | Reserved column |

| Column | Type | Description |
|--------------|--------------------------|--|
| rolauthinfo | Text | Additional information when LDAP authentication is used. If other authentication modes are used, the value is NULL . |
| rolpwdexpire | Integer | Password expiration time. Users can change their password before it expires. After the password expires, only the administrator can change the password. The value -1 indicates that the password never expires. |
| rolpwdtime | Timestamp with time zone | Time when a password is created |
| roluuid | Bigint | Role identifier. This column is available only in clusters of version 9.1.0 or later. |

18.2.16 PG_AUTH_HISTORY

PG_AUTH_HISTORY records the authentication history of the role. It is accessible only to users with system administrator rights.

Table 18-16 PG_AUTH_HISTORY columns

| Column | Type | Description |
|--------------|--------------------------|---|
| roloid | OID | Role identifier |
| passwordtime | Timestamp with time zone | Time of password creation and change |
| rolpassword | Text | Role password that is encrypted using MD5 or SHA256, or that is not encrypted |

18.2.17 PG_AUTH_MEMBERS

PG_AUTH_MEMBERS records the membership relations between roles.

Table 18-17 PG_AUTH_MEMBERS columns

| Column | Type | Description |
|---------|------|--|
| roleid | OID | ID of a role that has a member |
| member | OID | ID of a role that is a member of ROLEID |
| grantor | OID | ID of a role that grants this membership |

| Column | Type | Description |
|--------------|---------|---|
| admin_option | boolean | Whether a member can grant membership in ROLEID to others |

18.2.18 PG_BLOCKLISTS

PG_BLOCKLISTS records query filtering rules. This system catalog is supported only by clusters of version 9.1.0.100 or later.

Table 18-18 PG_BLOCKLISTS columns

| Column | Type | Description |
|------------------|--------------------------|--|
| block_name | Name | Name of a query filtering rule |
| role | OID | User OID bound to the query filtering rule |
| client_addr | inet | IP address of the client bound to the query filtering rule |
| application_name | Name | Name of the client bound to the query filtering rule |
| unique_sql_id | INT8 | unique_sql_id that matches the query filtering rule |
| sql_hash | Name | sql_hash that matches the query filtering rule |
| block_type | INT4 | Type of the statement bound to the query filtering rule. The type can be SELECT, UPDATE, INSERT, DELETE, or MERGE. |
| partition_num | INT4 | Estimated maximum number of partitions to be scanned |
| table_num | INT4 | Estimated maximum number of tables to be scanned |
| estimate_row | INT4 | Estimated maximum number of rows to be scanned |
| query_band | Text | Type of the job that is actively identified |
| sql | Text | SQL statement that matches the query filtering rule |
| created_time | Timestamp with time zone | Timestamp when a query filtering rule is created or modified |

| Column | Type | Description |
|----------------|---------|--|
| resource_pool | Name | Name of the resource pool to which the statement intercepted by the query filtering rule is switched. This column is available only in clusters of version 9.1.0.200 or later. |
| max_active_num | Integer | Maximum number of concurrent statements intercepted by the query filtering rule. If the value is lower than the specified limit, execution proceeds normally. However, if the value is equal to or exceeds the limit, an error is reported and the statements are intercepted.
This column is available only in clusters of version 9.1.0.200 or later. |
| is_warning | Boolean | Whether an error or alarm is reported when a statement is intercepted by the query filtering rule. <ul style="list-style-type: none">• false indicates that an error is reported when a statement is intercepted. The default value is false.• true indicates that an alarm is generated when a statement is intercepted. This column is available only in clusters of version 9.1.0.200 or later. |

18.2.19 PG_CAST

PG_CAST records conversion relationships between data types.

Table 18-19 PG_CAST columns

| Column | Type | Description |
|------------|------|--|
| castsource | OID | OID of the source data type |
| casttarget | OID | OID of the target data type |
| castfunc | OID | OID of the conversion function. If the value is 0 , no conversion function is required. |

| Column | Type | Description |
|-------------|------|---|
| castcontext | Char | Conversion mode between the source and target data types <ul style="list-style-type: none">• e indicates that only explicit conversion can be performed (using the CAST or :: syntax).• i indicates that only implicit conversion can be performed.• a indicates that both explicit and implicit conversion can be performed between data types. |
| castmethod | Char | Conversion method <ul style="list-style-type: none">• f indicates that conversion is performed using the specified function in the castfunc column.• b indicates that binary forcible conversion rather than the specified function in the castfunc column is performed between data types. |

18.2.20 PG_CLASS

PG_CLASS records database objects and their relations.

Table 18-20 PG_CLASS columns

| Column | Type | Description |
|--------------|------|--|
| OID | OID | Row identifier (hidden attribute; must be explicitly selected) |
| relname | Name | Name of an object, such as a table, index, or view |
| relnamespace | OID | OID of the namespace that contains the relationship |
| reltype | OID | Data type that corresponds to this table's row type (the index is 0 because the index does not have pg_type record) |
| reloftype | OID | OID is of composite type. 0 indicates other types. |
| relowner | OID | Owner of the relationship |
| relam | OID | Specifies the access method used, such as B-tree and hash, if this is an index |
| relfilenode | OID | Name of the on-disk file of this relationship. If such file does not exist, the value is 0 . |

| Column | Type | Description |
|----------------|------------------|---|
| reltablespace | OID | Tablespace in which this relationship is stored. If its value is 0 , the default tablespace in this database is used. This column is meaningless if the relationship has no on-disk file. |
| relpages | Double precision | Size of the on-disk representation of this table in pages (of size BLCKSZ). This is only an estimate used by the optimizer. |
| reltuples | Double precision | Number of rows in the table. This is only an estimate used by the optimizer. |
| relallvisible | Integer | Number of pages marked as all visible in the table. This column is used by the optimizer for optimizing SQL execution. It is updated by VACUUM , ANALYZE , and a few DDL statements such as CREATE INDEX . |
| reltoastrelid | OID | OID of the TOAST table associated with this table. The OID is 0 if no TOAST table exists.

The TOAST table stores large columns "offline" in a secondary table. |
| reltoastidxid | OID | OID of the index for a TOAST table. The OID is 0 for a table other than a TOAST table. |
| reldeltarelid | OID | OID of a Delta table

Delta tables belong to column-store tables. They store long tail data generated during data import. |
| reldeltaidx | OID | OID of the index for a Delta table |
| relcudescrelid | OID | OID of a CU description table

CU description tables (Desc tables) belong to column-store tables. They control whether storage data in the HDFS table directory is visible. |
| relcudescidx | OID | OID of the index for a CU description table |
| relhasindex | boolean | Its value is true if this column is a table and has (or recently had) at least one index.

It is set by CREATE INDEX but is not immediately cleared by DROP INDEX . If the VACUUM process detects that a table has no index, it clears the relhasindex column and sets the value to false . |
| relisshared | boolean | Its value is true if the table is shared across all databases in the cluster. Only certain system catalogs (such as pg_database) are shared. |

| Column | Type | Description |
|----------------|----------|--|
| relpersistence | Char | <ul style="list-style-type: none">• p indicates a permanent table.• u indicates a non-log table.• t indicates a local temporary table.• g indicates a global temporary table. |
| relkind | Char | <ul style="list-style-type: none">• r indicates an ordinary table.• i indicates an index.• S indicates a sequence.• v indicates a view.• c indicates the composite type.• t indicates a TOAST table.• f indicates a foreign table.• m indicates a materialized view. |
| relnatts | Smallint | Number of user columns in the relationship (excluding system columns) pg_attribute has the same number of rows corresponding to the user columns. |
| relchecks | Smallint | Number of constraints on a table. For details, see PG_CONSTRAINT . |
| relhasoids | boolean | Its value is true if an OID is generated for each row of the relationship. |
| relhaspkey | boolean | Its value is true if the table has (or once had) a primary key. |
| relhasrules | boolean | Its value is true if the table has rules. See table PG_REWRITE to check whether it has rules. |
| relhastriggers | boolean | Its value is true if the table has (or once had) triggers. For details, see PG_TRIGGER . |
| relhassubclass | boolean | Its value is true if the table has (or once had) any inheritance child table. |
| relcmprs | tinyint | Whether the compression feature is enabled for the table. Note that only batch insertion triggers compression so ordinary CRUD does not trigger compression. <ul style="list-style-type: none">• 0 indicates other tables that do not support compression (primarily system tables, on which the compression attribute cannot be modified).• 1 indicates that the compression feature of the table data is NOCOMPRESS or has no specified keyword.• 2 indicates that the compression feature of the table data is COMPRESS. |

| Column | Type | Description |
|------------------|-----------|--|
| relhasclusterkey | boolean | Whether the local cluster storage is used |
| relrowmovement | boolean | Whether the row migration is allowed when the partitioned table is updated <ul style="list-style-type: none">• true indicates that the row migration is allowed.• false indicates that the row migration is not allowed. |
| parttype | Char | Whether the table or index has the property of a partitioned table <ul style="list-style-type: none">• p indicates that the table or index has the property of a partitioned table.• n indicates that the table or index does not have the property of a partitioned table.• v indicates that the table is the value partitioned table in the HDFS. |
| relfrozenxid | xid32 | All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order to prevent transaction ID wraparound (or to allow pg_clog to be shrunk). The value is 0 (InvalidTransactionId) if the relationship is not a table.

To ensure forward compatibility, this column is reserved. The relfrozenxid64 column is added to record the information. |
| relacl | aclitem[] | Access permissions
The command output of the query is as follows:
rolename=xxxx/yyyy --Assigning privileges to a role
=xxxx/yyyy --Assigning the permission to public

xxxx indicates the assigned privileges, and yyyy indicates the roles that are assigned to the privileges. For details about permission descriptions, see Table 18-21 . |
| reloptions | text[] | Access-method-specific options, as "keyword=value" strings |
| relfrozenxid64 | Xid | All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order to prevent transaction ID wraparound (or to allow pg_clog to be shrunk). The value is 0 (InvalidTransactionId) if the relationship is not a table. |

Database object permissions are controlled using access control lists (ACLs). DWS uses **aclitem** to indicate the permissions on a specific database object.

- For databases and schemas, **aclitem** is stored in **pg_database.dataacl** and **pg_namespace.nspacl**.
- For other database objects such as tables and views, **aclitem** is stored in **pg_class.relacl**.
- For column-level permissions, **aclitem** is stored in **pg_attribute.attacl**.

For example, **test_user=a*r/test_user1** indicates that the **test_user** user has the INSERT and SELECT permissions on the current database object. The INSERT permission with **WITH GRANT OPTION**, means that other users can be granted the permission. **/test_user1** indicates that the **aclitem** permission is granted by **test_user1**.

The following table lists all database object permissions and their abbreviations in DWS.

Table 18-21 ACL permission parameters

| Permission | Abbreviation | Applicable Object Type |
|-----------------|--------------|--|
| SELECT (read) | r | LARGE OBJECT, SEQUENCE, TABLE (and table-like objects), and table column |
| UPDATE (write) | w | LARGE OBJECT, SEQUENCE, TABLE, and table column |
| INSERT (insert) | a | TABLE and table column |
| DELETE | d | TABLE |
| TRUNCATE | D | TABLE |
| REFERENCES | x | TABLE and table column |
| TRIGGER | t | TABLE |
| EXECUTE | X | FUNCTION and PROCEDURE |
| USAGE | U | DOMAIN, FOREIGN DATA WRAPPER, FOREIGN SERVER, LANGUAGE, SCHEMA, SEQUENCE, and TYPE |
| CREATE | C | DATABASE, SCHEMA, and TABLESPACE |
| CONNECT | c | DATABASE |
| TEMPORARY | T | DATABASE |
| ANALYZE ANALYSE | A | TABLE |
| ALTER | L | TABLE |
| DROP | P | TABLE |

| Permission | Abbreviation | Applicable Object Type |
|---|-------------------------|------------------------|
| VACUUM | v | TABLE |
| ALL PRIVILEGES | arwdDxt
A and
vLP | TABLE |
| Authorization options for preceding permissions | * | - |

Examples

View the OID and relfilenode of a table.

```
SELECT oid,relname,relfilenode FROM pg_class WHERE relname = 'table_name';
```

Count row-store tables.

```
SELECT 'row count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and  
reloptions::text not like '%column%' and reloptions::text not like '%internal_mask%';
```

Count column-store tables.

```
SELECT 'column count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and  
reloptions::text like '%column%';
```

Query the comments of all tables in the database.

```
SELECT relname as tablename,obj_description(relfilenode,'pg_class') as comment FROM pg_class;
```

Query the permission of a specified table in the database.

```
SELECT relacl FROM pg_class WHERE relname = 'table_name';
```

18.2.21 PG_COLLATION

PG_COLLATION records the available collations, which are essentially mappings from an SQL name to operating system locale categories.

Table 18-22 PG_COLLATION columns

| Name | Type | Reference | Description |
|---------------|------|-----------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| collname | Name | N/A | Collation name (unique per namespace and encoding) |
| collnamespace | OID | PG_NAMESPACE .oid | OID of the namespace that contains this collation |

| Name | Type | Reference | Description |
|--------------|---------|-------------------------------|---|
| collowner | OID | PG_AUTHID.oid | Owner of the collation |
| collencoding | Integer | N/A | Encoding in which the collation is applicable, or -1 if it works for any encoding |
| collcollate | Name | N/A | LC_COLLATE for this collation object |
| collctype | Name | N/A | LC_CTYPE for this collation object |

18.2.22 PG_CONSTRAINT

PG_CONSTRAINT records check, primary key, unique, and foreign key constraints on the tables.

Table 18-23 PG_CONSTRAINT columns

| Column | Type | Description |
|---------------|---------|--|
| conname | Name | Constraint name (not necessarily unique) |
| connamespace | OID | OID of the namespace that contains the constraint |
| contype | Char | <ul style="list-style-type: none">• c indicates check constraints.• f indicates foreign key constraints.• p indicates primary key constraints.• u indicates unique constraints.• t indicates trigger constraints. |
| condeferrable | boolean | Whether the constraint can be deferrable |
| condeferred | boolean | Whether the constraint can be deferrable by default |
| convalidated | boolean | Whether the constraint is valid Currently, only foreign key and check constraints can be set to false. |
| conrelid | OID | Table containing this constraint. The value is 0 if it is not a table constraint. |
| contypid | OID | Domain containing this constraint. The value is 0 if it is not a domain constraint. |
| conindid | OID | ID of the index associated with the constraint |
| confrelid | OID | Referenced table if this constraint is a foreign key; otherwise, the value is 0 . |

| Column | Type | Description |
|---------------|------------|---|
| confupdtype | Char | Foreign key update action code <ul style="list-style-type: none">• a indicates no action.• r indicates restriction.• c indicates cascading.• n indicates that the parameter is set to null.• d indicates that the default value is used. |
| confdeltype | Char | Foreign key deletion action code <ul style="list-style-type: none">• a indicates no action.• r indicates restriction.• c indicates cascading.• n indicates that the parameter is set to null.• d indicates that the default value is used. |
| confmatchtype | Char | Foreign key match type <ul style="list-style-type: none">• f indicates full match.• p indicates partial match.• u indicates simple match (not specified). |
| conislocal | boolean | Whether the local constraint is defined for the relationship |
| coninhcount | Integer | Number of direct inheritance parent tables this constraint has. When the number is not 0 , the constraint cannot be deleted or renamed. |
| connoinherit | boolean | Whether the constraint can be inherited |
| consoft | boolean | Whether the column indicates an informational constraint. |
| conopt | boolean | Whether you can use Informational Constraint to optimize the execution plan. |
| conkey | smallint[] | Column list of the constrained control if this column is a table constraint |
| confkey | smallint[] | List of referenced columns if this column is a foreign key |
| confpeqop | oid[] | ID list of the equality operators for PK = FK comparisons if this column is a foreign key |
| conppeqop | oid[] | ID list of the equality operators for PK = PK comparisons if this column is a foreign key |

| Column | Type | Description |
|-----------|--------------|---|
| conffeqop | oid[] | ID list of the equality operators for FK = FK comparisons if this column is a foreign key |
| conexclp | oid[] | ID list of the per-column exclusion operators if this column is an exclusion constraint |
| conbin | pg_node_tree | Internal representation of the expression if this column is a check constraint |
| consrc | Text | Human-readable representation of the expression if this column is a check constraint |

NOTICE

- **consrc** is not updated when referenced objects change; for example, it will not track renaming of columns. You are advised to use **pg_get_constraintdef ()** to extract the definition of a check constraint instead of depending on this column.
- **pg_class.relchecks** must be consistent with the number of check-constraint entries in this table for each relationship.

Example

Query whether a specified table has a primary key.

```
CREATE TABLE t1
(
  C_CUSTKEY  BIGINT      ,
  C_NAME     VARCHAR(25) ,
  C_ADDRESS  VARCHAR(40) ,
  C_NATIONKEY INT        ,
  C_PHONE    CHAR(15)    ,
  C_ACCTBAL  DECIMAL(15,2),
  CONSTRAINT C_CUSTKEY_KEY PRIMARY KEY(C_CUSTKEY,C_NAME)
)
DISTRIBUTE BY HASH(C_CUSTKEY,C_NAME);

SELECT conname FROM pg_constraint WHERE conrelid = 't1'::regclass AND contype = 'p';
 conname
-----
c_custkey_key
(1 row)
```

18.2.23 PG_CONVERSION

PG_CONVERSION records encoding conversion information.

Table 18-24 PG_CONVERSION columns

| Column | Type | Reference | Description |
|----------------|---------|----------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| conname | Name | N/A | Conversion name (unique in a namespace) |
| connamespace | OID | PG_NAMESPACE.oid | OID of the namespace that contains this conversion |
| conowner | OID | PG_AUTHID.oid | Owner of the conversion |
| conforencoding | Integer | N/A | Source encoding ID |
| contoencoding | Integer | N/A | Destination encoding ID |
| conproc | regproc | PG_PROC.oid | Conversion procedure |
| condefault | boolean | N/A | Whether the default conversion is used |

18.2.24 PG_DATABASE

PG_DATABASE records information about the available databases.

Table 18-25 PG_DATABASE columns

| Column | Type | Description |
|---------------|---------|---|
| datname | Name | Database name |
| datdba | OID | Owner of the database, usually the user who created it |
| encoding | Integer | Character encoding for this database
You can use <code>pg_encoding_to_char()</code> to convert this number to the encoding name. |
| datcollate | Name | Sequence used by the database |
| datctype | Name | Character type used by the database |
| datistemplate | boolean | Whether this column can serve as a template database |
| dataallowconn | boolean | If false then no one can connect to this database. This column is used to protect the template0 database from being altered. |

| Column | Type | Description |
|------------------|-----------|---|
| datconlimit | Integer | Maximum number of concurrent connections allowed on this database. -1 indicates no limit. |
| datlastsysoid | OID | Last system OID in the database |
| datfrozenxid | xid32 | Tracks whether the database needs to be vacuumed in order to prevent transaction ID wraparound.

To ensure forward compatibility, this column is reserved. The datfrozenxid64 column is added to record the information. |
| dattablespace | OID | Default tablespace of the database |
| datcompatibility | Name | Database compatibility mode <ul style="list-style-type: none">• ORA: compatible with the Oracle database• TD: compatible with the Teradata database• MySQL: compatible with the MySQL database |
| datacl | aclitem[] | Permission to access the database. |
| datfrozenxid64 | Xid | Tracks whether the database needs to be vacuumed in order to prevent transaction ID wraparound. |

18.2.25 PG_DB_ROLE_SETTING

PG_DB_ROLE_SETTING records the default values of configuration items bonded to each role and database when the database is running.

Table 18-26 PG_DB_ROLE_SETTING columns

| Column | Type | Description |
|-------------|--------|--|
| setdatabase | OID | Database corresponding to the configuration items; the value is 0 if the database is not specified. |
| setrole | OID | Role corresponding to the configuration items; the value is 0 if the role is not specified. |
| setconfig | text[] | Default value of configuration items when the database is running. |

18.2.26 PG_DEFAULT_ACL

PG_DEFAULT_ACL records the initial privileges assigned to the newly created objects.

Table 18-27 PG_DEFAULT_ACL columns

| Column | Type | Description |
|-----------------|-----------|---|
| defaclrole | OID | ID of the role associated with the permission |
| defaclnamespace | OID | Namespace associated with the permission; the value is 0 if no ID |
| defaclobjtype | Char | Object type of the permission: <ul style="list-style-type: none">• r indicates a table or view.• S indicates a sequence.• f indicates a function.• T indicates a type. |
| defaclacl | aclitem[] | Access permissions that this type of object should have on creation |

Examples

Run the following command to view the initial permissions of the new user **role1**:

```
select * from PG_DEFAULT_ACL;
defaclrole | defaclnamespace | defaclobjtype | defaclacl
-----+-----+-----+-----
16820 | 16822 | r | {role1=r/user1}
```

You can also run the following statement to convert the format:

```
SELECT pg_catalog.pg_get_userbyid(d.defaclrole) AS "Granter", n.nspname AS "Schema", CASE
d.defaclobjtype WHEN 'r' THEN 'table' WHEN 'S' THEN 'sequence' WHEN 'f' THEN 'function' WHEN 'T'
THEN 'type' END AS "Type", pg_catalog.array_to_string(d.defaclacl, E', ') AS "Access privileges" FROM
pg_catalog.pg_default_acl d LEFT JOIN pg_catalog.pg_namespace n ON n.oid = d.defaclnamespace ORDER
BY 1, 2, 3;
```

If the following information is displayed, **user1** grants **role1** the read permission on schema **user1**.

```
Granter | Schema | Type | Access privileges
-----+-----+-----+-----
user1 | user1 | table | role1=r/user1
(1 row)
```

18.2.27 PG_DEPEND

PG_DEPEND records the dependency relationships between database objects. This information allows **DROP** commands to find which other objects must be dropped by **DROP CASCADE** or prevent dropping in the **DROP RESTRICT** case.

See also [PG_SHDEPEND](#), which provides similar functionality for recording dependencies between objects that are shared between database clusters.

Table 18-28 PG_DEPEND columns

| Column | Type | Reference | Description |
|-------------|---------|------------------------------|--|
| classid | OID | PG_CLASS.oid | OID of the system catalog the dependent object is in |
| objid | OID | Any OID column | OID of the specific dependent object |
| objsubid | Integer | N/A | For a table column, this is the column number (the objid and classid refer to the table itself). For all other object types, this column is 0 . |
| refclassid | OID | PG_CLASS.oid | OID of the system catalog the referenced object is in |
| refobjid | OID | Any OID column | OID of the specific referenced object |
| refobjsubid | Integer | N/A | For a table column, this is the column number (the refobjid and refclassid refer to the table itself). For all other object types, this column is 0 . |
| deptype | Char | N/A | A code defining the specific semantics of this dependency relationship |

In all cases, a **pg_depend** entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors defined by **deptype**:

- **DEPENDENCY_NORMAL (n)**: A normal relationship between separately-created objects. The dependent object can be dropped without affecting the referenced object. The referenced object can only be dropped by specifying **CASCADE**, in which case the dependent object is dropped, too. Example: a table column has a normal dependency on its data type.
- **DEPENDENCY_AUTO (a)**: The dependent object can be dropped separately from the referenced object, and should be automatically dropped (regardless of **RESTRICT** or **CASCADE** mode) if the referenced object is dropped. Example: a named constraint on a table is made autodependent on the table, so that it will go away if the table is dropped.
- **DEPENDENCY_INTERNAL (i)**: The dependent object was created as part of creation of the referenced object, and is only a part of its internal implementation. A **DROP** of the dependent object will be disallowed outright (We'll tell the user to issue a **DROP** against the referenced object, instead). A **DROP** of the referenced object will be propagated through to drop the dependent object whether **CASCADE** is specified or not. For example, a trigger used to enforce a foreign key constraint is set to an item internally dependent on its constraint in [PG_CONSTRAINT](#).
- **DEPENDENCY_EXTENSION (e)**: The dependent object is a member of the extension that is the referenced object. (For details, see [PG_EXTENSION](#)).

dependent object can be dropped via **DROP EXTENSION** on the referenced object. Functionally this dependency type acts the same as an internal dependency, but it is kept separate for clarity and to simplify **gs_dump**.

- **DEPENDENCY_PIN (p)**: There is no dependent object. This indicates that the system itself depends on the referenced object, and therefore the object cannot be deleted. Entries of this type are created only by **initdb**. The columns with dependent object are all zeroes.

Examples

Query the table that depends on the database object sequence **serial1**:

1. Query the OID of the sequence **serial1** in the system catalog **PG_CLASS**.

```
SELECT oid FROM pg_class WHERE relname = 'serial1';
OID
-----
17815
(1 row)
```

2. Use the system catalog **PG_DEPEND** and the OID of **serial1** to obtain the objects that depend on **serial1**.

```
SELECT * FROM pg_depend WHERE objid = '17815';
classid | objid | objsubid | refclassid | refobjid | refobjsubid | deptype
-----+-----+-----+-----+-----+-----+-----
1259 | 17815 | 0 | 2615 | 2200 | 0 | n
1259 | 17815 | 0 | 1259 | 17812 | 1 | a
(2 rows)
```

3. Obtain the OID of the table that depends on the serial1 sequence based on the refobjid field and query the table name. The result indicates that the table **customer_address** depends on **serial1**.

```
SELECT relname FROM pg_class where oid = '17812';
relname
-----
customer_address
(1 row)
```

18.2.28 PG_DESCRIPTION

PG_DESCRIPTION records optional descriptions (comments) for each database object. Descriptions of many built-in system objects are provided in the initial contents of **PG_DESCRIPTION**.

See also **PG_SHDESCRIPTION**, which performs a similar function for descriptions involving objects that are shared across a database cluster.

Table 18-29 PG_DESCRIPTION columns

| Column | Type | Reference | Description |
|----------|------|-------------------------------|--|
| objoid | OID | Any OID column | OID of the object this description pertains to |
| classoid | OID | PG_CLASS .oid | OID of the system catalog this object appears in |

| Column | Type | Reference | Description |
|-------------|---------|-----------|---|
| objsubid | Integer | - | For a comment on a table column, this is the column number (the objoid and classoid refer to the table itself). For all other object types, this column is 0 . |
| description | Text | - | Arbitrary text that serves as the description of this object |

18.2.29 PG_ENUM

PG_ENUM records entries showing the values and labels for each enum type. The internal representation of a given enum value is actually the OID of its associated row in **pg_enum**.

Table 18-30 PG_ENUM columns

| Column | Type | Reference | Description |
|---------------|------|-----------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| enumtypid | OID | PG_TYPE.oid | OID of pg_type that contains this enum value |
| enumsortorder | Real | N/A | Sort position of this enum value within its enum type |
| enumlabel | Name | N/A | Textual label for this enum value |

The OIDs for **PG_ENUM** rows follow a special rule: even-numbered OIDs are guaranteed to be ordered in the same way as the sort ordering of their enum type. That is, if two even OIDs belong to the same enum type, the smaller OID must have the smaller **enumsortorder** value. Odd-numbered OID values need bear no relationship to the sort order. This rule allows the enum comparison routines to avoid catalog lookups in many common cases. The routines that create and alter enum types attempt to assign even OIDs to enum values whenever possible.

When an enum type is created, its members are assigned sort-order positions from 1 to *n*. But members added later might be given negative or fractional values of **enumsortorder**. The only requirement on these values is that they be correctly ordered and unique within each enum type.

18.2.30 PG_EXCEPT_RULE

The **PG_EXCEPT_RULE** system catalog stores information about exception rules. An exception rule set consists of multiple exception rules with the same name.

Table 18-31 PG_EXCEPT_RULE

| Column | Type | Description |
|--------|------|--|
| Name | Name | Name of an exception rule set. |
| rule | Name | Type of a rule in the exception rule set, or action taken when the current exception rule set is triggered. (For example, it can be blocktime , elapsedtime , spillsize , or an action taken after an exception rule is triggered.) |
| value | Name | Rule threshold corresponding to the exception rule. If it specifies the action taken after an exception rule is triggered, its value is abort . |

18.2.31 PG_EXTERNAL_NAMESPACE

Stores EXTERNAL SCHEMA information. This system catalog is supported only in 8.3.0 and later versions.

Table 18-32 PG_EXTERNAL_NAMESPACE columns

| Name | Type | Description |
|------------|--------|---|
| nspid | OID | External schema OID |
| srvname | Text | Name of the foreign server |
| source | Text | Metadata service type |
| address | Text | Metadata service address |
| database | Text | Metadata server database |
| confpath | Text | Path of the configuration file of the metadata server |
| ensoptions | Text[] | Reserved column, which is left empty currently. |
| catalog | Text | Metadata server catalog |

Example

Query the created **EXTERNAL SCHEMA ex1**:

```
SELECT * FROM pg_external_namespace WHERE nspid = (SELECT oid FROM pg_namespace WHERE  
nspname = 'ex1');
```

18.2.32 PG_EXTENSION

PG_EXTENSION records information about the installed extensions. By default, DWS has 34 extensions: aio_scheduler, btree_gin, cudesckv, dimsearch, dist_fdw,

functional_clog, functional_extension, functional_file, functional_hudi, functional_job, functional_largeobject, functional_memory, functional_other, functional_signal, functional_vacuum, gc_fdw, hdfs_fdw, hstore, log_fdw, operational_backup, operational_cgroup, operational_cudesc, operational_other, operational_replication, operational_restoration, operational_stats, operational_xlog, packages, pgcrypto, pldbgapi, plpgsql, roach_api, tsdb, and uuid-oss.

Table 18-33 PG_EXTENSION

| Column | Type | Description |
|----------------|---------|---|
| extname | Name | Extension name |
| extowner | OID | Owner of the extension |
| extnamespace | OID | Namespace containing the extension's exported objects |
| extrelocatable | boolean | Whether the extension can be relocated to another schema |
| extversion | Text | Version number of the extension |
| extconfig | oid[] | Configuration information about the extension |
| extcondition | Text[] | Filter conditions for the extension's configuration information |

18.2.33 PG_EXTENSION_DATA_SOURCE

PG_EXTENSION_DATA_SOURCE records information about external data source. An external data source contains information about an external database, such as its password encoding. It is mainly used with Extension Connector.

Table 18-34 PG_EXTENSION_DATA_SOURCE columns

| Name | Type | Reference | Description |
|------------|------|----------------|--|
| OID | OID | - | Row identifier (hidden attribute; must be explicitly selected) |
| srcname | Name | - | Name of an external data source |
| srcowner | OID | PG_AUTH ID.oid | Owner of an external data source |
| srctype | Text | - | Type of an external data source. It is NULL by default. |
| srcversion | Text | - | Type of an external data source. It is NULL by default. |

| Name | Type | Reference | Description |
|------------|-----------|-----------|---|
| srcacl | aclitem[] | - | Access permissions |
| srcoptions | Text[] | - | Option used for foreign data sources. It is a keyword=value string. |

18.2.34 PG_FINE_DR_INFO

The **PG_FINE_DR_INFO** system catalog records the replay status of the fine-grained DR standby table. This system catalog is supported only by clusters of version 8.2.0.100 or later.

Table 18-35 PG_FINE_DR_INFO columns

| Name | Type | Description |
|---------------|--------------------------|--|
| OID | OID | Row identifier (hidden attribute; displayed only when explicitly selected) |
| relid | OID | OID of the standby fine-grained DR table |
| lastcsn | Xid | CSN of the last successful playback |
| lastxmin | Xid | xmin of the last successful playback |
| lastxmax | Xid | xmax of the last successful playback |
| laststarttime | Timestamp with time zone | Start time of the last successful playback |
| lastendtime | Timestamp with time zone | End time of the last successful playback |

Examples

Check the playback status of the standby table in the DR cluster.

```
SELECT * FROM pg_fine_dr_info;  
relid | lastcsn | lastxmin | lastxmax | laststarttime | lastendtime  
-----+-----+-----+-----+-----+-----  
21132 | 1251610 | 1251609 | 1251611 | 2023-01-04 20:51:58.375136+08 | 2023-01-04 20:51:58.393986+08  
(1 row)
```

18.2.35 PG_FOREIGN_DATA_WRAPPER

PG_FOREIGN_DATA_WRAPPER records foreign-data wrapper definitions. A foreign-data wrapper is the mechanism by which external data, residing on foreign servers, is accessed.

Table 18-36 PG_FOREIGN_DATA_WRAPPER columns

| Name | Type | Reference | Description |
|--------------|-----------|-------------------------------|---|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| fdwname | Name | N/A | Name of the foreign-data wrapper |
| fdwowner | OID | PG_AUTHID.oid | Owner of the foreign-data wrapper |
| fdwhandler | OID | PG_PROC.oid | References a handler function that is responsible for supplying execution routines for the foreign-data wrapper. Its value is 0 if no handler is provided. |
| fdwvalidator | OID | PG_PROC.oid | References a validator function that is responsible for checking the validity of the options given to the foreign-data wrapper, as well as options for foreign servers and user mappings using the foreign-data wrapper. Its value is 0 if no validator is provided. |
| fdwacl | aclitem[] | N/A | Access permissions |
| fdwoptions | Text[] | N/A | Option used for foreign data wrappers. It is a keyword=value string. |

18.2.36 PG_FOREIGN_SERVER

PG_FOREIGN_SERVER records the foreign server definitions. A foreign server describes a source of external data, such as a remote server. Foreign servers are accessed via foreign-data wrappers.

Table 18-37 PG_FOREIGN_SERVER columns

| Name | Type | Reference | Description |
|----------|------|---|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| srvname | Name | N/A | Name of the foreign server |
| srvowner | OID | PG_AUTHID.oid | Owner of the foreign server |
| srvfdw | OID | PG_FOREIGN_DATA_WRAPPER.oid | OID of the foreign-data wrapper of this foreign server |

| Name | Type | Reference | Description |
|------------|-----------|-----------|--|
| srvtype | Text | N/A | Type of the server (optional) |
| srvversion | Text | N/A | Version of the server (optional) |
| srvacl | aclitem[] | N/A | Access permissions |
| srvoptions | Text[] | N/A | Option used for foreign servers. It is a keyword=value string. |

18.2.37 PG_FOREIGN_TABLE

PG_FOREIGN_TABLE records auxiliary information about foreign tables.

Table 18-38 PG_FOREIGN_TABLE columns

| Column | Type | Description |
|-------------|---------|--|
| ftrelid | OID | OID of the foreign table |
| ftserver | OID | OID of the server where the foreign table is located |
| ftwriteonly | boolean | Whether data can be written in the foreign table |
| ftoptions | Text[] | Foreign table options |

18.2.38 PG_INDEX

PG_INDEX records part of the information about indexes. The rest is mostly in **PG_CLASS**.

Table 18-39 PG_INDEX columns

| Column | Type | Description |
|-------------|----------|--|
| indexrelid | OID | OID of the pg_class entry for this index |
| indrelid | OID | OID of the pg_class entry for the table this index is for |
| indnatts | Smallint | Number of columns in an index |
| indisunique | boolean | This index is a unique index if the value is true . |

| Column | Type | Description |
|----------------|------------|---|
| indisprimary | boolean | This index represents the primary key of the table if the value is true . If this value is true , the value of indisunique is true. |
| indisexclusion | boolean | This index supports exclusion constraints if the value is true . |
| indimmediate | boolean | A uniqueness check is performed upon data insertion if the value is true . |
| indisclustered | boolean | The table was last clustered on this index if the value is true . |
| indisusable | boolean | This index supports insert/select if the value is true . |
| indisvalid | boolean | This index is valid for queries if the value is true . If this column is false , this index is possibly incomplete and must still be modified by INSERT/UPDATE operations, but it cannot safely be used for queries. If it is a unique index, the uniqueness property is also not true. |
| indcheckxmin | boolean | If the value is true , queries must not use the index until the xmin of this row in pg_index is below their TransactionXmin event horizon, because the table may contain broken HOT chains with incompatible rows that they can see. |
| indisready | boolean | If the value is true , this index is ready for inserts. If the value is false , this index is ignored when data is inserted or modified. |
| indkey | int2vector | This is an array of indnatts values that indicate which table columns this index creates. For example, a value of 1 3 means that the first and the third columns make up the index key. 0 in this array indicates that the corresponding index attribute is an expression over the table columns, rather than a simple column reference. |
| indcollation | oidvector | ID of each column used by the index |
| indclass | oidvector | For each column in the index key, this column contains the OID of the operator class to use. For details, see PG_OPCLASS . |
| indoption | int2vector | Array of values that store per-column flag bits. The meaning of the bits is defined by the index's access method. |

| Column | Type | Description |
|-------------------|--------------|---|
| indexprs | pg_node_tree | Expression trees (in nodeToString() representation) for index attributes that are not simple column references. It is a list with one element for each zero entry in INDKEY . NULL if all index attributes are simple references. |
| indpred | pg_node_tree | Expression tree (in nodeToString() representation) for partial index predicate. If the index is not a partial index, the value is null. |
| indnullstreatment | tinyint | <p>Processing mode of the NULL value in the unique index. This field is valid only if indisunique is set to true.</p> <p>Options:</p> <ul style="list-style-type: none">• 0: NULLS DISTINCT. NULL values are not equivalent and can be inserted repeatedly.• 1: NULLS NOT DISTINCT. NULL values are equivalent and cannot be inserted repeatedly.• 2: NULLS IGNORE. NULL columns are ignored during equivalent comparison. If all index columns are NULL, NULL values can be inserted repeatedly. If part of the index columns are NULL, data can be inserted only if non-null values are different. <p>Default value: 0</p> <p>NOTE</p> <ul style="list-style-type: none">• If the current cluster was upgraded from an earlier version to 8.2.0.100, the value of this field is NULL for existing indexes. For newly created indexes, the value of this field is determined by the [NULLS [NOT] DISTINCT NULLS IGNORE] field. The default value is 0.• If the current cluster is newly installed and its version is 8.2.0.100, for newly created indexes, the value of this field is determined by the [NULLS [NOT] DISTINCT NULLS IGNORE] field. The default value is 0. |

18.2.39 PG_INHERITS

PG_INHERITS records information about table inheritance hierarchies. There is one entry for each direct child table in the database. Indirect inheritance can be determined by following chains of entries.

Table 18-40 PG_INHERITS columns

| Column | Type | Reference | Description |
|-----------|---------|------------------------------|--|
| inhrelid | OID | PG_CLASS.oid | OID of the child table. |
| inhparent | OID | PG_CLASS.oid | OID of the parent table. |
| inhseqno | Integer | - | If there is more than one direct parent for a child table (multiple inheritances), this number tells the order in which the inherited columns are to be arranged. The count starts at 1. |

18.2.40 PG_JOB_INFO

PG_JOB_INFO records the execution results of scheduled tasks. The schema of the system catalog is **dbms_om**.

Table 18-41 dbms_om.pg_job_info columns

| Column | Type | Description |
|------------|---------------------|---------------------------------------|
| job_id | Integer | Job ID |
| job_db | OID | OID of the database where the task is |
| start_time | Timestamp with zone | Task execution start time |
| status | character(8) | Task execution status |
| end_time | Timestamp with zone | Task execution end time |
| err_msg | Text | Task execution error information |

18.2.41 PG_JOBS

PG_JOBS records detailed information about jobs created by users. Dedicated threads poll the **pg_jobs** table and trigger jobs based on scheduled job execution time. This table belongs to the Shared Relation category. All job records are visible to all databases.

Table 18-42 PG_JOBS columns

| Name | Type | Description |
|--------|---------|---|
| job_id | Integer | Job ID, primary key, unique (with a unique index) |

| Name | Type | Description |
|-----------------|-----------------------------|--|
| what | Text | Job content |
| log_user | OID | Username of the job creator |
| priv_user | OID | User ID of the job executor |
| job_db | OID | OID of the database where the job is executed |
| job_nsp | OID | OID of the namespace where a job is running |
| job_node | OID | CN node on which the job will be created and executed |
| is_broken | Boolean | Whether the current job is invalid |
| start_date | Timestamp without time zone | Start time of the first job execution, accurate to millisecond |
| next_run_date | Timestamp without time zone | Scheduled time of the next job execution, accurate to millisecond |
| failure_count | Smallint | Number of consecutive failures |
| interval | Text | Job execution interval |
| last_start_date | Timestamp without time zone | Start time of the last job execution, accurate to millisecond |
| last_end_date | Timestamp without time zone | End time of the last job execution, accurate to millisecond |
| last_suc_date | Timestamp without time zone | Start time of the last successful job execution, accurate to millisecond |
| this_run_date | Timestamp without time zone | Start time of the ongoing job execution, accurate to millisecond |

18.2.42 PG_LANGUAGE

PG_LANGUAGE records languages that can be used to write functions or stored procedures.

Table 18-43 PG_LANGUAGE columns

| Name | Type | Reference | Description |
|---------------|-----------|-------------------------------|---|
| OID | OID | N/A | Row identifier (hidden attribute; must be explicitly selected) |
| lanname | Name | N/A | Name of the language |
| lanowner | OID | PG_AUTHID.oid | Owner of the language |
| lanispl | boolean | N/A | The value is false for internal languages (such as SQL) and true for user-defined languages. Currently, gs_dump still uses this to determine which languages need to be dumped, but this might be replaced by a different mechanism in the future. |
| lanpltrusted | boolean | N/A | Its value is true if this is a trusted language, which means that it is believed not to grant access to anything outside the normal SQL execution environment. Only the initial user can create functions in untrusted languages. |
| lanplcallfoid | OID | PG_AUTHID.oid | For external languages, this references the language handler, which is a special function that is responsible for executing all functions that are written in the particular language. |
| laninline | OID | PG_AUTHID.oid | This references a function that is responsible for executing "inline" anonymous code blocks (DO blocks). The value is 0 if inline blocks are not supported. |
| lanvalidator | OID | PG_AUTHID.oid | This references a language validator function that is responsible for checking the syntax and validity of new functions when they are created. The value is 0 if no validator is provided. |
| lanacl | aclitem[] | N/A | Access permissions |

18.2.43 PG_LARGEOBJECT

PG_LARGEOBJECT records the data making up large objects. A large object is identified by an OID assigned when it is created. Each large object is broken into

segments or "pages" small enough to be conveniently stored as rows in **pg_largeobject**. The amount of data per page is defined to be LOBLKSIZE (which is currently BLCKSZ/4, or typically 2 kB).

It is accessible only to users with system administrator rights.

Table 18-44 PG_LARGEOBJECT columns

| Column | Type | Reference | Description |
|--------|---------|---|--|
| loid | OID | PG_LARGEOBJECT_METADATA.oid | Identifier of the large object that includes this page. |
| pageno | Integer | - | Page number of this page within its large object (counting from zero). |
| data | bytea | - | Actual data stored in the large object. This will never be more than LOBLKSIZE bytes and might be less. |

Each row of **pg_largeobject** holds data for one page of a large object, beginning at byte offset (**pageno * LOBLKSIZE**) within the object. The implementation allows sparse storage: pages might be missing, and might be shorter than **LOBLKSIZE** bytes even if they are not the last page of the object. Missing regions within a large object are read as 0.

18.2.44 PG_LARGEOBJECT_METADATA

PG_LARGEOBJECT_METADATA records metadata associated with large objects. The actual large object data is stored in **PG_LARGEOBJECT**.

Table 18-45 PG_LARGEOBJECT_METADATA columns

| Name | Type | Reference | Description |
|----------|-----------|-------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| lomowner | OID | PG_AUTHID.oid | Owner of the large object |
| lomacl | aclitem[] | N/A | Access permissions |

18.2.45 PG_MATVIEW

PG_MATVIEW records materialized view information about the current node.

Table 18-46 PG_MATVIEW columns

| Column | Type | Description |
|---------------------|------------|---|
| mvid | OID | OID of the materialized view. |
| build_mode | Char | Build mode of the materialized view. <ul style="list-style-type: none">• 'd': indicates "deferred", which means that data is contained in the materialized view only when the view is refreshed for the first time.• 'i': indicates "immediate", which means that the latest data is included when the materialized view is created. |
| refresh_method | Char | Refresh method of the materialized view.
'c': indicates complete refresh. |
| refresh_mode | Char | Refresh mode of the materialized view.
'd': stands for demand, indicating on-demand update. |
| rewrite_enable | Boolean | Indicates whether query rewriting of the materialized view is supported. |
| active | Boolean | Indicates whether the materialized view needs to be refreshed. |
| relnum | Int | Number of materialized view base tables. |
| start_time | timestampz | Time when the materialized view is refreshed for the first time. If this parameter is left blank, the first refresh time is the current time plus the interval. |
| interval | Interval | Interval for refreshing the materialized view. |
| refresh_time | timestampz | Last refresh time of the materialized view. |
| refresh_finish_time | timestampz | End time of the last refresh of a materialized view. |

18.2.46 PG_MATVIEW_DEPEND

PG_MATVIEW_DEPEND records the base tables used in materialized views, along with the corresponding materialized views for each base table. This view is supported only by clusters of version 8.2.1.100 or later.

Table 18-47 PG_MATVIEW_DEPEND columns

| Column | Type | Description |
|--------------------|-----------|--|
| matviewid | OID | OID of the materialized view. |
| relid | OID | OID of the materialized view base table. |
| partids | oidvector | OID of the base table partition when the partition is specified. |
| contain_entire_rel | Boolean | Whether to include the entire base table. |
| matpartid | OID | Partition OID of the mapped materialized view. |
| relpartid | OID | Partition OID of the mapped base table. |

18.2.47 PG_MATVIEW_PARTITION

PG_MATVIEW_PARTITION records data information about each partition in partitioned materialized views. This system catalog is supported only by clusters of version 9.1.0.200 or later.

Table 18-48 PG_MATVIEW_PARTITION columns

| Column | Type | Description |
|---------------------|--------------------------|---|
| matpartid | OID | OID of the materialized view partition. |
| active | Boolean | Whether the partition data is the most up-to-date data.
t indicates that the data in the partitioned materialized view is in the latest state.
f indicates that the data in the partitioned materialized view is not in the latest state. |
| refresh_start_time | Timestamp with time zone | Start time of the last refresh. |
| refresh_finish_time | Timestamp with time zone | End time of the last refresh. |

18.2.48 PG_NAMESPACE

PG_NAMESPACE records the namespaces, that is, schema-related information.

Table 18-49 PG_NAMESPACE columns

| Column | Type | Description |
|-------------|-----------|--|
| nspname | Name | Name of the namespace |
| nspowner | OID | Owner of the namespace |
| nsptimeline | Bigint | Timeline when the namespace is created on the DN This column is for internal use and valid only on the DN. |
| nspacl | aclitem[] | Access permissions For details, see sections "GRANT" and "REVOKE" in <i>Data Warehouse Service (DWS) SQL Syntax Reference</i> . |
| permSPACE | Bigint | Quota of a schema's permanent tablespace |
| usedSPACE | Bigint | Used size of a schema's permanent tablespace |
| nsptype | Char | Distinguishes external schemas from common schemas. This parameter is supported only in 8.3.0 and later versions to adapt to LakeFormation features.
NOTE
The nsptype field is added to distinguish external schemas from common schemas. <ul style="list-style-type: none">• e indicates an external schema.• i indicates a common schema. |

18.2.49 PG_OBJECT

PG_OBJECT records the user creation, creation time, last modification time, and last analyzing time of objects of specified types (types existing in **object_type**).

Table 18-50 PG_OBJECT columns

| Column | Type | Description |
|-------------|------|--|
| object_oid | OID | Object identifier. |
| object_type | Char | Object type: <ul style="list-style-type: none">• r indicates a table, which can be an ordinary table or a temporary table.• i indicates an index.• s indicates a sequence.• v indicates a view.• p indicates a stored procedure and function.• f indicates a foreign table. |
| creator | OID | ID of the creator. |

| Column | Type | Description |
|-------------------|--------------------------|---|
| ctime | Timestamp with time zone | Object creation time. |
| mtime | Timestamp with time zone | Time when the object was last modified. By default, the ALTER , COMMENT , GRANT/REVOKE , and TRUNCATE operations are recorded.

object_mtime_record_mode can be used to control whether ALTER , COMMENT , GRANT/REVOKE , and TRUNCATE operations are recorded. |
| last_analyze_time | Timestamp with time zone | Time when an object is analyzed for the last time. |

NOTICE

- Only normal user operations are recorded. Operations before the object upgrade and during the **initdb** process cannot be recorded.
- **ctime** and **mtime** are the start time of the transaction.
- The time of object modification due to capacity expansion is also recorded.

18.2.50 PG_OBSSCANINFO

PG_OBSSCANINFO defines the OBS runtime information scanned in cluster acceleration scenarios. Each record corresponds to a piece of runtime information of a foreign table on OBS in a query.

Table 18-51 PG_OBSSCANINFO columns

| Name | Type | Reference | Description |
|-------------|--------------------------|-----------|---|
| query_id | Bigint | N/A | Query ID |
| user_id | Text | N/A | Database user who performs queries |
| table_name | Text | N/A | Name of a foreign table on OBS |
| file_type | Text | N/A | Format of files storing the underlying data |
| time_stamp | timestamp with time zone | N/A | Scanning start time |
| actual_time | Double | N/A | Scanning execution time, in seconds |

| Name | Type | Reference | Description |
|--------------|--------|-----------|--------------------------------|
| file_scanned | Bigint | N/A | Number of files scanned |
| data_size | Double | N/A | Size of data scanned, in bytes |
| billing_info | Text | N/A | Reserved column |

18.2.51 PG_OPCLASS

PG_OPCLASS defines index access method operator classes.

Each operator class defines semantics for index columns of a particular data type and a particular index access method. An operator class essentially specifies that a particular operator family is applicable to a particular indexable column data type. The set of operators from the family that are actually usable with the indexed column are whichever ones accept the column's data type as their lefthand input.

Table 18-52 PG_OPCLASS columns

| Name | Type | Reference | Description |
|--------------|---------|----------------------------------|---|
| OID | OID | - | Row identifier (hidden attribute; must be explicitly selected) |
| opcmethod | OID | PG_AM.oid | Index access method the operator class is for |
| opcname | Name | - | Name of the operator class |
| opcnamespace | OID | PG_NAMESPACE.oid | Namespace to which the operator class belongs |
| opcowner | OID | PG_AUTHID.oid | Owner of the operator class |
| opcfamily | OID | PG_OPFAMILY.oid | Operator family containing the operator class |
| opcintype | OID | PG_TYPE.oid | Data type that the operator class indexes |
| opcdefault | boolean | - | Whether the operator class is the default for opcintype . If it is, its value is true . |
| opckeytype | OID | PG_TYPE.oid | Type of data stored in index, or zero if same as opcintype |

An operator class's **opcmethod** must match the **opfmetho**d of its containing operator family. Also, there must be no more than one **pg_opclass** row having **opcdefault** true for any given combination of **opcmethod** and **opcintype**.

18.2.52 PG_OPERATOR

PG_OPERATOR records information about operators.

Table 18-53 PG_OPERATOR columns

| Name | Type | Reference | Description |
|--------------|---------|----------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| oprname | Name | N/A | Name of the operator |
| oprnamespace | OID | PG_NAMESPACE.oid | OID of the namespace that contains this operator |
| oprowner | OID | PG_AUTHID.oid | Owner of the operator |
| oprkind | Char | N/A | <ul style="list-style-type: none">• b: infix ("both")• l: prefix ("left")• r: postfix ("right") |
| oprcanmerge | boolean | N/A | Whether the operator supports merge joins |
| oprcanhash | boolean | N/A | Whether the operator supports hash joins |
| oprleft | OID | PG_TYPE.oid | Type of the left operand |
| oprright | OID | PG_TYPE.oid | Type of the right operand |
| oprresult | OID | PG_TYPE.oid | Type of the result |
| oprcom | OID | PG_OPERATOR.oid | Commutator of this operator, if any |
| oprnegate | OID | PG_OPERATOR.oid | Negator of this operator, if any |
| oprcode | regproc | PG_PROC.oid | Function that implements this operator |
| oprrest | regproc | PG_PROC.oid | Restriction selectivity estimation function for this operator |
| oprjoin | regproc | PG_PROC.oid | Join selectivity estimation function for this operator |

18.2.53 PG_OPFAMILY

PG_OPFAMILY defines operator families.

Each operator family is a collection of operators and associated support routines that implement the semantics specified for a particular index access method.

Furthermore, the operators in a family are all "compatible", in a way that is specified by the access method. The operator family concept allows cross-data-type operators to be used with indexes and to be reasoned about using knowledge of access method semantics.

Table 18-54 PG_OPFAMILY columns

| Name | Type | Reference | Description |
|--------------|------|-----------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| opfmethod | OID | PG_AM .oid | Index method used by the operator family |
| opfname | Name | N/A | Name of the operator family |
| opfnamespace | OID | PG_NAMESPACE .oid | Namespace of the operator family |
| opfowner | OID | PG_AUTHID .oid | Owner of the operator family |

The majority of the information defining an operator family is not in **PG_OPFAMILY**, but in the associated [PG_AMOP](#), [PG_AMPROC](#), and [PG_OPCLASS](#).

18.2.54 PG_PARTITION

PG_PARTITION records all partitioned tables, table partitions, toast tables on table partitions, and index partitions in the database. Partitioned index information is not stored in the **PG_PARTITION** system catalog.

Table 18-55 PG_PARTITION columns

| Column | Type | Description |
|----------|------|--|
| relname | Name | Names of the partitioned tables, table partitions, TOAST tables on table partitions, and index partitions |
| parttype | Char | Object type <ul style="list-style-type: none">• r indicates a partitioned table.• p indicates a table partition.• x indicates an index partition.• t indicates a TOAST table. |

| Column | Type | Description |
|----------------|------------------|---|
| parentid | OID | OID of the partitioned table in PG_CLASS when the object is a partitioned table or table partition
OID of the partitioned index when the object is an index partition |
| rangenum | Integer | Reserved field. |
| intervalnum | Integer | Reserved field. |
| partstrategy | Char | Partition policy of the partitioned table. Only the following policies are supported:
r indicates the range partition.
v indicates the numeric partition.
l indicates the list partition. |
| relfilenode | OID | Physical storage locations of the table partition, index partition, and TOAST table on the table partition. |
| reltablespace | OID | OID of the tablespace containing the table partition, index partition, TOAST table on the table partition |
| relpages | Double precision | Statistics: numbers of data pages of the table partition and index partition |
| reltuples | Double precision | Statistics: numbers of tuples of the table partition and index partition |
| relallvisible | Integer | Statistics: number of visible data pages of the table partition and index partition |
| reltoastrelid | OID | OID of the TOAST table corresponding to the table partition |
| reltoastidxid | OID | OID of the TOAST table index corresponding to the table partition |
| indextblid | OID | OID of the table partition corresponding to the index partition |
| indisusable | boolean | Whether the index partition is available |
| reldeltarelid | OID | OID of a Delta table |
| reldeltaidx | OID | OID of the index for a Delta table |
| relcudescrelid | OID | OID of a CU description table |
| relcudescidx | OID | OID of the index for a CU description table |

| Column | Type | Description |
|--------------------|--------------|--|
| relfrozenxid | xid32 | Frozen transaction ID
To ensure forward compatibility, this column is reserved. The relfrozenxid64 column is added to record the information. |
| intspnum | Integer | Number of tablespaces that the interval partition belongs to |
| partkey | int2vector | Column number of the partition key |
| intervaltablespace | oidvector | Tablespace that the interval partition belongs to. Interval partitions fall in the tablespaces in the round-robin manner. |
| interval | Text[] | Interval value of the interval partition |
| boundaries | Text[] | Upper boundary of the range partition and interval partition |
| transit | Text[] | Transit of the interval partition |
| reloptions | Text[] | Storage property of a partition used for collecting online scale-out information. Same as pg_class.reloptions , it is a keyword=value string. |
| relfrozenxid64 | Xid | Frozen transaction ID |
| boundexprs | pg_node_tree | Partition boundary expression. <ul style="list-style-type: none">For range partitioning, it is the upper boundary expression of a partition.For list partitioning, it is a collection of partition boundary enumeration values. The pg_node_tree data is not readable. You can use the expression pg_get_expr to translate the current column into readable information.
<pre>SELECT pg_get_expr(boundexprs, 0) FROM pg_partition WHERE relname = 'country_202201'; pg_get_expr ----- ROW(202201, 'city1'::text), ROW(202201, 'city2'::text) (1 row)</pre> |
| relmetaversion | Xid | Metadata version information. This column is supported only by clusters of version 9.1.0 or later. |

Example

Query the partition information of the partitioned table **web_returns_p2**.

```
CREATE TABLE web_returns_p2
(
```

```
wr_returned_date_sk    integer,
wr_returned_time_sk    integer,
wr_item_sk             integer NOT NULL,
wr_refunded_customer_sk integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
    PARTITION p2016 START(20161231) END(20191231) EVERY(10000),
    PARTITION p0 END(maxvalue)
);

SELECT oid FROM pg_class WHERE relname = 'web_returns_p2';
OID
-----
97628

SELECT relname,parttype,parentid,boundaries FROM pg_partition WHERE parentid = '97628';
 relname  | parttype | parentid | boundaries
-----+-----+-----+-----
web_returns_p2 | r       | 97628 | 
p2016_0      | p       | 97628 | {20161231}
p2016_1      | p       | 97628 | {20171231}
p2016_2      | p       | 97628 | {20181231}
p2016_3      | p       | 97628 | {20191231}
p0           | p       | 97628 | {NULL}
(6 rows)
```

18.2.55 PG_PLTEMPLATE

PG_PLTEMPLATE records template information for procedural languages.

Table 18-56 PG_PLTEMPLATE columns

| Column | Type | Description |
|---------------|-----------|---|
| tmplname | Name | Name of the language for which this template is used. |
| tmpltrusted | boolean | The value is true if the language is considered trusted. |
| tmpldbacreate | boolean | The value is true if the language is created by the owner of the database. |
| tmplhandler | Text | Name of the call handler function. |
| tmplinline | Text | Name of the anonymous block handler. If no name of the block handler exists, the value is null. |
| tmplvalidator | Text | Name of the verification function. If no verification function is available, the value is null. |
| tmpllibrary | Text | Path of the shared library that implements languages. |
| tmplacl | aclitem[] | Access permissions for template (not yet used). |

18.2.56 PG_PROC

PG_PROC records information about functions or procedures.

Table 18-57 PG_PROC columns

| Column | Type | Description |
|--------------|---------|---|
| proname | Name | Name of the function |
| pronamespace | OID | OID of the namespace that contains the function |
| proowner | OID | Owner of the function |
| prolang | OID | Implementation language or call interface of the function |
| procost | Real | Estimated execution cost |
| prorows | Real | Estimate number of result rows |
| provariadic | OID | Data type of parameter element |
| protransform | regproc | Simplified call method for this function |
| proisagg | boolean | Whether this function is an aggregate function |
| proiswindow | boolean | Whether this function is a window function |
| prosecdef | boolean | Whether this function is a security definer (such as a "setuid" function) |
| proleakproof | boolean | Whether this function has side effects. If no leakproof treatment is provided for parameters, the function throws errors. |
| proisstrict | boolean | The function returns null if any call parameter is null. In that case the function does not actually be called at all. Functions that are not "strict" must be prepared to process null inputs. |
| proretset | boolean | The function returns a set, that is, multiple values of the specified data type. |
| provolatile | Char | Whether the function's result depends only on its input parameters, or is affected by outside factors <ul style="list-style-type: none">It is i for "immutable" functions, which always deliver the same result for the same inputs.It is s for "stable" functions, whose results (for fixed inputs) do not change within a scan.It is v for "volatile" functions, whose results may change at any time. |

| Column | Type | Description |
|-----------------|--------------|--|
| pronargs | Smallint | Number of parameters |
| pronargdefaults | Smallint | Number of parameters that have default values |
| prorettype | OID | OID of the returned parameter type |
| proargtypes | oidvector | Array with the data types of the function parameters. This array includes only input parameters (including INOUT parameters) and thus represents the call signature of the function. |
| proallargtypes | oid[] | Array with the data types of the function parameters. This array includes all parameter types (including OUT and INOUT parameters); however, if all the parameters are IN parameters, this column is null. Note that array subscripting is 1-based, whereas for historical reasons, and proargtypes is subscripted from 0. |
| proargmodes | "char"[] | Array with the modes of the function parameters. <ul style="list-style-type: none">• i indicates IN parameters.• o indicates OUT parameters.• b indicates INOUT parameters.• v indicates VARIADIC parameters.• t indicates table-valued parameters. If all the parameters are IN parameters, this column is null. Note that subscripts of this array correspond to positions of proallargtypes not proargtypes . |
| proargnames | Text[] | Array that stores the names of the function parameters. Parameters without a name are set to empty strings in the array. If none of the parameters have a name, this column is null. Note that subscripts correspond to positions of proallargtypes not proargtypes . |
| proargdefaults | pg_node_tree | Expression tree of the default value. This is the list of PRONARGDEFAULTS elements. |
| prosrc | Text | A definition that describes a function or stored procedure. In an interpreting language, it is the function source code, a link symbol, a file name, or any body content specified when a function or stored procedure is created, depending on how a language or calling is used. |

| Column | Type | Description |
|------------------|------------|---|
| probin | Text | Additional information about how to call the function. Again, the interpretation is language-specific. |
| proconfig | Text[] | Function's local settings for run-time configuration variables. |
| proacl | aclitem[] | Access permissions For details, see sections "GRANT" and "REVOKE" in <i>Data Warehouse Service (DWS) SQL Syntax Reference</i> . |
| prodefaultargpos | int2vector | Locations of the function default values. Not only the last few parameters have default values. |
| fencedmode | boolean | Execution mode of a function, indicating whether a function is executed in fence or not fence mode. If the execution mode is fence, the function is executed in the fork process that is reworked. The default value is fence . |
| proshippable | boolean | Whether a function can be pushed down to DNs. The default value is false . <ul style="list-style-type: none">• Functions of the IMMUTABLE type can always be pushed down to the DNs.• Functions of the STABLE or VOLATILE type can be pushed down to DNs only if their attribute is SHIPPABLE. |
| propackage | boolean | Indicates whether the function supports overloading, which is mainly used for the Oracle style function. The default value is false . |

Examples

Query the OID of a specified function. For example, obtain the OID **1295** of the **justify_days** function.

```
SELECT oid FROM pg_proc where proname = 'justify_days';
OID
-----
1295
(1 row)
```

Query whether a function is an aggregate function. For example, the **justify_days** function is a non-aggregate function.

```
SELECT proisagg FROM pg_proc where proname = 'justify_days';
proisagg
-----
f
(1 row)
```

18.2.57 PG_PUBLICATION

PG_PUBLICATION records all the publications created in the current database. This system catalog is supported only by clusters of version 8.2.0.100 or later.

Table 18-58 PG_PUBLICATION columns

| Column | Type | Reference | Description |
|--------------|---------|-------------------------------|---|
| OID | OID | - | Row identifier (hidden attribute; displayed only when explicitly selected) |
| pubname | Name | - | Publication name |
| pubowner | OID | PG_AUTHID.oid | Publication owner |
| puballtables | Boolean | - | If its value is true , the publication includes all the tables in the database, including any tables that will be created in the future. |
| pubinsert | Boolean | - | If its value is true , the INSERT operation is copied for the tables in the publication. |
| pubupdate | Boolean | - | If its value is true , the UPDATE operation is copied for the tables in the publication. |
| pubdelete | Boolean | - | If its value is true , the DELETE operation is copied for the tables in the publication. |
| pubtruncate | Boolean | - | If its value is true , the TRUNCATE operation is copied for the tables in the publication. |

Examples

View all releases.

```
SELECT * FROM pg_publication;
pubname | pubowner | puballtables | pubinsert | pubupdate | pubdelete | pubtruncate
-----+-----+-----+-----+-----+-----+-----
mypub   | 10      | t           | t         | t         | t         | t
(1 row)
```

18.2.58 PG_PUBLICATION_NAMESPACE

PG_PUBLICATION_NAMESPACE records the mapping between publications and schemas in the current database, which is a many-to-many mapping. This system catalog is supported only by clusters of version 8.2.0.100 or later.

Table 18-59 PG_PUBLICATION_NAMESPACE columns

| Name | Type | Reference | Description |
|---------|------|------------------------------------|--|
| OID | OID | - | Row identifier (hidden attribute; displayed only when explicitly selected) |
| prpubid | OID | PG_PUBLICATION.oid | Publication OID in the mapping |
| pnnspid | OID | PG_NAMESPACE.oid | Schema OID in the mapping |

Examples

View all mappings between publications and schemas.

```
SELECT * FROM pg_publication_namespace;  
pnpubid | pnnspid  
-----+-----  
16797 | 16796  
(1 row)
```

18.2.59 PG_PUBLICATION_REL

PG_PUBLICATION_REL records the mapping between publications and tables in the current database, which is a many-to-many mapping. This system catalog is supported only by clusters of version 8.2.0.100 or later.

NOTE

To check detailed information, you are advised to use the [PG_PUBLICATION_TABLES](#) view.

Table 18-60 PG_PUBLICATION_REL columns

| Name | Type | Reference | Description |
|---------|------|------------------------------------|--|
| OID | OID | - | Row identifier (hidden attribute; displayed only when explicitly selected) |
| prpubid | OID | PG_PUBLICATION.oid | Publication OID in the mapping |
| prrelid | OID | PG_CLASS.oid | OID of the mapped table |

Examples

View all mappings between publications and tables.

```
postgres=# SELECT * FROM pg_publication_rel;  
prpubid | prrelid  
-----+-----  
16797 | 16757  
16797 | 16776  
(2 rows)
```

18.2.60 PG_RANGE

PG_RANGE records information about range types.

This is in addition to the types' entries in [PG_TYPE](#).

Table 18-61 PG_RANGE columns

| Column | Type | Reference | Description |
|--------------|---------|-----------------------------------|---|
| rngtypid | OID | PG_TYPE .oid | OID of the range type |
| rngsubtype | OID | PG_TYPE .oid | OID of the element type (subtype) of this range type |
| rngcollation | OID | PG_COLLATION .oid | OID of the collation used for range comparisons, or 0 if none |
| rngsubopc | OID | PG_OPCLASS .oid | OID of the subtype's operator class used for range comparisons |
| rngcanonical | regproc | PG_PROC .oid | OID of the function to convert a range value into canonical form, or 0 if none |
| rngsubdiff | regproc | PG_PROC .oid | OID of the function to return the difference between two element values as double precision , or 0 if none |

rngsubopc (plus **rngcollation**, if the element type is collatable) determines the sort ordering used by the range type. **rngcanonical** is used when the element type is discrete.

18.2.61 PG_REDACTION_COLUMN

PG_REDACTION_COLUMN records the information about the redacted columns.

Table 18-62 PG_REDACTION_COLUMN columns

| Column | Type | Description |
|---------------|----------|---------------------------------------|
| object_oid | OID | OID of the object to be redacted. |
| column_attrno | Smallint | attrno of the redacted column. |

| Column | Type | Description |
|------------------------|--------------|---|
| function_type | Integer | Redaction type.
NOTE
This column is reserved. It is used only for forward compatibility of redacted column information in earlier versions. The value can be 0 (NONE) or 1 (FULL). |
| function_parameters | Text | Parameters used when the redaction type is partial (reserved). |
| regexp_pattern | Text | Pattern string when the redaction type is regexp (reserved). |
| regexp_replace_string | Text | Replacement string when the redaction type is regexp (reserved). |
| regexp_position | Integer | Start and end replacement positions when the redaction type is regexp (reserved). |
| regexp_occurrence | Integer | Replacement times when the redaction type is regexp (reserved). |
| regexp_match_parameter | Text | Regular control parameter used when the redaction type is regexp (reserved). |
| column_description | Text | Description of the redacted column. |
| function_expr | pg_node_tree | Internal representation of the redaction function. |
| inherited | bool | Whether a redacted column is inherited from another redacted column. |

| Column | Type | Description |
|------------|------|--|
| policy_oid | OID | OID of the masking policy.
Supported by clusters of 8.2.1.100 and later versions. It is used to search for masked column information from the metadata in the system catalog. |

18.2.62 PG_REDACTION_POLICY

PG_REDACTION_POLICY records information about the object to be redacted.

Table 18-63 PG_REDACTION_POLICY columns

| Column | Type | Description |
|--------------------|--------------|--|
| object_oid | OID | OID of the object to be redacted. |
| policy_name | Name | Name of the redaction policy. |
| enable | Boolean | Policy status (enabled or disabled).
NOTE
The value can be: <ul style="list-style-type: none">• true: enabled• false: disabled |
| expression | pg_node_tree | Policy effective expression (for users). |
| policy_description | Text | Description of a policy. |
| inherited | Bool | Whether a redaction policy is inherited from another redaction policy. |
| policy_order | float4 | Masking policy sequence. This field is supported by 8.2.1.100 and later cluster versions. |

18.2.63 PG_RELFILENODE_SIZE

The **PG_RELFILENODE_SIZE** system catalog provides file-level space statistics. Each record in the table corresponds to a physical file on the disk and the size of the file.

Table 18-64 PG_RELFILENODE_SIZE columns

| Column | Type | Description |
|--------------|---------|---|
| databaseid | OID | OID of the database that the physical file belongs to. If a system catalog is shared across databases, its value is 0 . |
| tablespaceid | OID | Tablespace OID of the physical file |
| relfilenode | OID | Serial number of the physical file |
| backendid | Integer | ID of the background thread that creates the physical file. Generally, the value is -1 . |
| type | Integer | Type of the physical file. <ul style="list-style-type: none">• The value 0 indicates a data file.• The value 1 indicates an FSM file.• The value 2 indicates a VM file.• The value 3 indicates a BCM file.• If the value greater than 4 indicates the total size of the data file and BCM file of the column in a column-store table. |
| filesize | Bigint | Size of the physical file, in bytes. |

18.2.64 PG_RLSPOLICY

PG_RLSPOLICY displays the information about row-level access control policies.

Table 18-65 PG_RLSPOLICY columns

| Column | Type | Description |
|----------|------|--|
| polname | Name | Name of a row-level access control policy |
| polrelid | OID | Table OID of a row-level access control policy |
| polcmd | Char | SQL operations affected by a row-level access control policy. The options are *(ALL) , r(SELECT) , w(UPDATE) , and d(DELETE) . |

| Column | Type | Description |
|---------------|--------------|--|
| polpermissive | Boolean | Type of a row-level access control policy
NOTE
Values of polpermissive : <ul style="list-style-type: none">• true: The row-level access control policy is a permissive policy.• false: The row-level access control policy is a restrictive policy. |
| polroles | oid[] | OID of database user affected by a row-level access control policy |
| polqual | pg_node_tree | SQL condition expression of a row-level access control policy |

18.2.65 PG_RESOURCE_POOL

PG_RESOURCE_POOL records the information about database resource pool.

Table 18-66 PG_RESOURCE_POOL columns

| Column | Type | Description |
|-------------------|---------|---|
| respool_name | Name | Name of the resource pool |
| mem_percent | Integer | Configured memory percentage. 0 indicates that the memory of the resource pool is not controlled. |
| cpu_affinity | Bigint | Reserved column without an actual meaning |
| control_group | Name | Name of the Cgroup where the resource pool is located |
| active_statements | Integer | Maximum number of concurrent statements in the resource pool |
| max_dop | Integer | Maximum number of concurrent simple jobs allowed by the resource pool. -1 and 0 indicate that there are no limitations. |
| memory_limit | Name | Estimated memory upper limit for a query. |
| parentid | OID | OID of the parent resource pool |
| io_limits | Integer | Reserved column without an actual meaning |
| io_priority | Text | Reserved column without an actual meaning |
| nodegroup | Name | Name of the logical cluster associated with the resource pool. The value is installation for a non-logical cluster. |

| Column | Type | Description |
|-------------|---------|---|
| is_foreign | Boolean | Whether the resource pool can be used for users outside the logical cluster. If it is set to true , the resource pool controls the resources of common users who do not belong to the current resource pool. |
| short_acc | Boolean | Whether to enable short query acceleration for a resource pool. This function is enabled by default. <ul style="list-style-type: none">• If short query acceleration is enabled, simple queries are controlled on the fast lane.• If short query acceleration is disabled, and simple queries are controlled on the slow lane. |
| except_rule | Text | Exception rule associated with a resource pool. There can be multiple associated rules, which are separated by commas (,). |
| weight | Integer | Resource scheduling weight. Currently, this parameter is used only for network scheduling. |

18.2.66 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 18-67 PG_REWRITE columns

| Column | Type | Description |
|----------|----------|---|
| rulename | Name | Name of the rule |
| ev_class | OID | Name of the table that uses the rule |
| ev_attr | Smallint | Column this rule is for (always 0 to indicate the entire table) |
| ev_type | Char | Event type for this rule: <ul style="list-style-type: none">• 1 = SELECT• 2 = UPDATE• 3 = INSERT• 4 = DELETE |

| Column | Type | Description |
|--------------|--------------------------|--|
| ev_enabled | Char | Controls in which mode the rule fires <ul style="list-style-type: none">• O: The rule fires in "origin" and "local" modes.• D: The rule is disabled.• R: The rule fires in "replica" mode.• A: The rule always fires. |
| is_instead | boolean | Its value is true if the rule is an INSTEAD rule. |
| ev_qual | pg_node_tree | Expression tree (in the form of a nodeToString() representation) for the rule's qualifying condition |
| ev_action | pg_node_tree | Query tree (in the form of a nodeToString() representation) for the rule's action |
| state_change | Timestamp with time zone | Time when the ev_enabled field is updated. This column is available only in clusters of version 9.1.0.200 or later. |

18.2.67 PG_SECLABEL

PG_SECLABEL records security labels on database objects.

See also [PG_SHSECLABEL](#), which performs a similar function for security labels of database objects that are shared across a database cluster.

Table 18-68 PG_SECLABEL columns

| Name | Type | Reference | Description |
|----------|---------|-------------------------------|--|
| objoid | OID | Any OID column | OID of the object this security label pertains to |
| classoid | OID | PG_CLASS .oid | OID of the system catalog that contains the object |
| objsubid | Integer | N/A | For a security label on a table column, this is the column number. |
| provider | Text | N/A | Label provider associated with this label |
| label | Text | N/A | Security label applied to this object |

18.2.68 PG_SHDEPEND

PG_SHDEPEND records the dependency relationships between database objects and shared objects, such as roles. This information allows DWS to ensure that those objects are unreferenced before attempting to delete them.

See also **PG_DEPEND**, which performs a similar function for dependencies involving objects within a single database.

Unlike most system catalogs, **PG_SHDEPEND** is shared across all databases of a cluster: there is only one copy of **PG_SHDEPEND** per cluster, not one per database.

Table 18-69 PG_SHDEPEND columns

| Name | Type | Reference | Description |
|------------|---------|----------------------------------|--|
| dbid | OID | PG_DATABASE .oid | OID of the database the dependent object is in. The value is 0 for a shared object. |
| classid | OID | PG_CLASS .oid | OID of the system catalog the dependent object is in. |
| objid | OID | Any OID column | OID of the specific dependent object |
| objsubid | Integer | - | For a table column, this is the column number (the objid and classid refer to the table itself). For all other object types, this column is 0 . |
| refclassid | OID | PG_CLASS .oid | OID of the system catalog the referenced object is in (must be a shared catalog) |
| refobjid | OID | Any OID column | OID of the specific referenced object |
| deptype | Char | - | Code segment defining the specific semantics of this dependency relationship. See the following text for details. |
| objfile | Text | - | Path of the user-defined C function library file. |

In all cases, a **pg_shdepend** entry indicates that the referenced object cannot be dropped without also dropping the dependent object. However, there are several subflavors defined by **deptype**:

- **SHARED_DEPENDENCY_OWNER** (o)
The referenced object (which must be a role) is the owner of the dependent object.

- **SHARED_DEPENDENCY_ACL** (a)
The referenced object (which must be a role) is mentioned in the ACL (access control list, i.e., privileges list) of the dependent object. (A **SHARED_DEPENDENCY_ACL** entry is not made for the owner of the object, since the owner will have a **SHARED_DEPENDENCY_OWNER** entry anyway.)
- **SHARED_DEPENDENCY_PIN** (p)
There is no dependent object. This type of entry is a signal that the system itself depends on the referenced object, and so that object must never be deleted. Entries of this type are created only by **initdb**. The columns for the dependent object contain zeroes.

18.2.69 PG_SHDESCRIPTION

PG_SHDESCRIPTION records optional comments for shared database objects. Descriptions can be manipulated with the **COMMENT** command and viewed with `gsq!'`s `\d` commands.

See also **PG_DESCRIPTION**, which performs a similar function for descriptions involving objects within a single database.

Unlike most system catalogs, **PG_SHDESCRIPTION** is shared across all databases of a cluster. There is only one copy of **PG_SHDESCRIPTION** per cluster, not one per database.

Table 18-70 PG_SHDESCRIPTION columns

| Name | Type | Reference | Description |
|-------------|------|-------------------------------|--|
| objoid | OID | Any OID column | OID of the object this description pertains to |
| classoid | OID | PG_CLASS .oid | OID of the system catalog where the object resides |
| description | Text | N/A | Arbitrary text that serves as the description of this object |

18.2.70 PG_SHSECLABEL

PG_SHSECLABEL records security labels on shared database objects. Security labels can be manipulated with the **SECURITY LABEL** command.

For an easier way to view security labels, see [PG_SECLABELS](#).

See also [PG_SECLABEL](#), which performs a similar function for security labels involving objects within a single database.

Unlike most system catalogs, **PG_SHSECLABEL** is shared across all databases of a cluster. There is only one copy of **PG_SHSECLABEL** per cluster, not one per database.

Table 18-71 PG_SHSECLABEL columns

| Name | Type | Reference | Description |
|----------|------|------------------------------|--|
| objoid | OID | Any OID column | OID of the object this security label pertains to |
| classoid | OID | PG_CLASS.oid | OID of the system catalog where the object resides |
| provider | Text | N/A | Label provider associated with this label |
| label | Text | N/A | Security label applied to this object |

18.2.71 PG_STATISTIC

PG_STATISTIC records statistics about tables and index columns in a database. It is accessible only to users with system administrator rights.

Table 18-72 PG_STATISTIC columns

| Column | Type | Description |
|-------------|----------|--|
| starelid | OID | Table or index which the described column belongs to. |
| starelkind | Char | Type of an object. |
| staattnum | Smallint | Number of the described column in the table, starting from 1. |
| stainherit | boolean | Whether to collect statistics for objects that have inheritance relationship. |
| stanullfrac | Real | Percentage of column entries that are null. |
| stawidth | Integer | Average stored width, in bytes, of non-null entries. |
| stadistinct | Real | Number of distinct, not-null data values in the column for all DNs. <ul style="list-style-type: none">• A value greater than zero is the actual number of distinct values.• A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.)• 0 indicates that the number of distinct values is unknown. |
| stakindN | Smallint | Code number stating that the type of statistics is stored in Slot N of the pg_statistic row.
Value range: 1 to 5 |

| Column | Type | Description |
|---------------|----------|--|
| staopN | OID | Operator used to generate the statistics stored in Slot N. For example, a histogram slot shows the < operator that defines the sort order of the data.
Value range: 1 to 5 |
| stanumbersN | real[] | Numerical statistics of the appropriate type for Slot N. The value is null if the slot kind does not involve numerical values.
Value range: 1 to 5 |
| stavaluesN | anyarray | Column data values of the appropriate type for Slot N. The value is null if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray.
Value range: 1 to 5 |
| stadndistinct | Real | Number of unique non-null data values in the dn1 column. <ul style="list-style-type: none">• A value greater than zero is the actual number of distinct values.• A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.)• 0 indicates that the number of distinct values is unknown. |
| staextinfo | Text | Information about extension statistics (reserved) |

18.2.72 PG_STATISTIC_DATA

Records extended statistics of tables in the current cluster, including expression statistics. This system catalog is supported only by clusters of version 8.2.1.200 or later.

Table 18-73 PG_STATISTIC_DATA columns

| Column | Type | Description |
|---------|------|--|
| statoid | OID | OID of the object for which extended statistics are collected. |

| Column | Type | Description |
|--------------|-----------------|--|
| inherit | Boolean | Whether to collect statistics for objects that have inheritance relationship. <ul style="list-style-type: none">• true indicates that the table has child tables.• false indicates that the table does not have child tables. |
| ndistinct | pg_ndistinct | N-distinct count, which is serialized as the pg_ndistinct type. |
| dependencies | pg_dependencies | Function dependency statistics, which are serialized as the pg_dependencies type. |
| mcv | pg_mcv_list | Most common value (MCV) statistics, which are serialized as the pg_mcv_list type. |
| expr | pg_statistic[] | Statistics of each expression, which is serialized as an array of the pg_statistic type. |

18.2.73 PG_STATISTIC_EXT

PG_STATISTIC_EXT records extended statistics about tables in a database. The range of extended statistics to be collected is specified by users. Only system administrators can access this system catalog.

Table 18-74 PG_STATISTIC_EXT columns

| Column | Type | Description |
|-------------|---------|---|
| starelid | OID | Table or index which the described column belongs to |
| starelkind | Char | Type of an object |
| stainherit | boolean | Whether to collect statistics for objects that have inheritance relationship |
| stanullfrac | Real | Percentage of column entries that are null |
| stawidth | Integer | Average stored width, in bytes, of non-null entries |
| stadistinct | Real | Number of distinct, not-null data values in the column for all DNs <ul style="list-style-type: none">• A value greater than zero is the actual number of distinct values.• A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.)• 0 indicates that the number of distinct values is unknown. |

| Column | Type | Description |
|---------------|--------------|---|
| stadndistinct | Real | Number of unique non-null data values in the dn1 column <ul style="list-style-type: none">A value greater than zero is the actual number of distinct values.A value less than zero is the negative of a multiplier for the number of rows in the table. (For example, stadistinct=-0.5 indicates that values in a column appear twice on average.)0 indicates that the number of distinct values is unknown. |
| stakindN | Smallint | Code number stating that the type of statistics is stored in Slot N of the pg_statistic row.
Value range: 1 to 5 |
| staopN | OID | Operator used to generate the statistics stored in Slot N. For example, a histogram slot shows the < operator that defines the sort order of the data.
Value range: 1 to 5 |
| stakey | int2vector | Array of a column ID |
| stanumbersN | real[] | Numerical statistics of the appropriate type for Slot N. The value is null if the slot kind does not involve numerical values.
Value range: 1 to 5 |
| stavaluesN | anyarray | Column data values of the appropriate type for Slot N. The value is null if the slot type does not store any data values. Each array's element values are actually of the specific column's data type so there is no way to define these columns' type more specifically than anyarray.
Value range: 1 to 5 |
| staexprs | pg_node_tree | Expression corresponding to the extended statistics information. |

18.2.74 PG_STATISTIC_INFO

Records extended statistics objects of tables in the current cluster, including expression statistics objects. This system catalog is supported only by clusters of version 8.2.1.200 or later.

Table 18-75 PG_STATISTIC_INFO columns

| Column | Type | Description |
|---------------|--------------|--|
| relid | OID | Table used in the extended statistics object. |
| statname | Name | Name of the extended statistics object. |
| statnamespace | OID | OID of the namespace that contains the extended statistics object. |
| statowner | OID | OID of the owner of the extended statistics object. |
| stattarget | Integer | Controls the detail level of statistics. |
| keys | int2vector | Array of column numbers described, indicating the columns included in the statistics object. |
| kind | "char"[] | Array containing statistics types. The possible values are as follows:
d : N-distinct statistics
f : function dependency statistics
m : most common value (MCV) statistics
e : expression statistics |
| exprs | pg_node_tree | Expression corresponding to the extended statistics information. |

18.2.75 PG_STAT_OBJECT

Records table statistics and autovacuum efficiency information of the current DB instance, and creates indexes for the **databaseid**, **relid**, and **partid** columns. Update of this system catalog is controlled by the [enable_pg_stat_object](#) parameter. This system catalog is supported only by clusters of version 8.2.1 or later.

Table 18-76 PG_STAT_OBJECT columns

| Column | Type | Reference | Description |
|------------|--------|----------------------------------|---|
| databaseid | OID | PG_DATABASE.oid | Database OID. |
| relid | OID | PG_CLASS.oid | Table OID. It is the OID of the primary table for a partitioned table. |
| partid | OID | PG_PARTITION.oid | Partition OID. If the table is not partitioned, the value is 0 . |
| numscans | Bigint | N/A | Number of times that sequential scans are started. |

| Column | Type | Reference | Description |
|------------------------------------|--------------------------|-----------|--|
| tuples_returned | Bigint | N/A | Number of visible tuples fetched by sequential scans. |
| tuples_fetched | Bigint | N/A | Number of visible tuples fetched. |
| tuples_inserted | Bigint | N/A | Number of inserted records. |
| tuples_updated | Bigint | N/A | Number of updated records. |
| tuples_deleted | Bigint | N/A | Number of deleted records. |
| tuples_hot_updated | Bigint | N/A | Number of HOT updates. |
| n_live_tuples | Bigint | N/A | Number of visible tuples. |
| last_autovacuum_begin_n_dead_tuple | Bigint | N/A | Number of tuples deleted before Autovacuum is executed. |
| n_dead_tuples | Bigint | N/A | Number of tuples deleted after Autovacuum is successful. |
| changes_since_analyze | Bigint | N/A | Last data modification time after Analyze. |
| blocks_fetched | Bigint | N/A | Number of selected pages. |
| blocks_hit | Bigint | N/A | Number of scanned pages. |
| cu_mem_hit | Bigint | N/A | Number of CU memory hits. |
| cu_hdd_sync | Bigint | N/A | Times that CUs are synchronously read from disks. |
| cu_hdd_async | Bigint | N/A | Times that CUs are asynchronously read from disks. |
| data_changed_timestamp | Timestamp with time zone | N/A | Last data modification time. |
| data_access_timestamp | Timestamp with time zone | N/A | Last access time of a table. |
| analyze_timestamp | Timestamp with time zone | N/A | Last Analyze time. |
| analyze_count | Bigint | N/A | Total number of Analyze times. |

| Column | Type | Reference | Description |
|------------------------------|--------------------------|-----------|---|
| autovac_analyze_timestamp | Timestamp with time zone | N/A | Last Autoanalyze time. |
| autovac_analyze_count | Bigint | N/A | Total number of Autoanalyze times. |
| vacuum_timestamp | Timestamp with time zone | N/A | Time of the latest Vacuum. |
| vacuum_count | Bigint | N/A | Total number of Vacuum times. |
| autovac_vacuum_timestamp | Timestamp with time zone | N/A | Last Autovacuum time. |
| autovac_vacuum_count | Bigint | N/A | Total number of Autovacuum times. |
| autovacuum_success_count | Bigint | N/A | Total number of successful Autovacuum operations. |
| last_autovacuum_time_cost | Bigint | N/A | Time spent on the latest successful Autovacuum, in microseconds. |
| avg_autovacuum_time_cost | Bigint | N/A | Average execution time of successful Autovacuum operations. Unit: μ s. |
| last_autovacuum_failed_count | Bigint | N/A | Total number of autovacuum failures since the last successful Autovacuum. |
| last_autovacuum_trigger | Smallint | N/A | Triggering mode of the latest autovacuum, which helps maintenance personnel determine the Vacuum status. |
| last_autovacuum_oldestxmin | Bigint | N/A | oldestxmin after the latest successful Autovacuum execution. If the table-level oldestxmin feature is enabled, this field records the value of oldestxmin used by the latest (AUTO)VACUUM of the table. |
| last_autovacuum_scan_pages | Bigint | N/A | Number of pages last scanned by autovacuum (only for row-store tables). |

| Column | Type | Reference | Description |
|--------------------------------------|--------|-----------|--|
| last_autovacuum_dirty_pages | Bigint | N/A | Number of pages last modified by Autovacuum (only for row-store tables). |
| last_autovacuum_clear_dead_tuples | Bigint | N/A | Number of dead tuples last cleared by Autovacuum (only for row-store tables) |
| sum_autovacuum_scan_pages | Bigint | N/A | Total number of pages scanned by Autovacuum since database initialization (only for row-store tables). |
| sum_autovacuum_dirty_pages | Bigint | N/A | Number of pages modified by Autovacuum since database initialization (only for row-store tables). |
| sum_autovacuum_clear_dead_tuples | Bigint | N/A | Total number of dead tuples cleared by Autovacuum since database initialization (only for row-store tables). |
| last_autovacuum_begin_cu_size | Bigint | N/A | Size of the CU file before the latest Autovacuum operation (only for column-store tables) |
| last_autovacuum_cu_size | Bigint | N/A | Size of the CU file after the latest Autovacuum (only for column-store tables) |
| last_autovacuum_rewrite_size | Bigint | N/A | Size of the column-store file last rewritten by autovacuum (only for column-store tables). |
| last_autovacuum_clear_size | Bigint | N/A | Size of the column-store file last cleared by Autovacuum (only for column-store tables). |
| last_autovacuum_clear_cbtuple_tuples | Bigint | N/A | Number of cbtree tuples last cleared by Autovacuum (only for column-store tables) |
| sum_autovacuum_rewrite_size | Bigint | N/A | Total size of column-store files rewritten by Autovacuum since database initialization (only for column-store tables). |
| sum_autovacuum_clear_size | Bigint | N/A | Total size of column-store files cleared by Autovacuum since database initialization (only for column-store tables). |

| Column | Type | Reference | Description |
|------------------------------------|--------------------------|-----------|---|
| sum_autovacuum_clear_cbtree_tuples | Bigint | N/A | Total number of cbtree tuples cleared by Autovacuum since database initialization (only for column-store tables). |
| last_autovacuum_csn | Bigint | N/A | If the table-level oldestxmin feature is enabled, this field records the CSN value corresponding to the latest oldestxmin value used by the table (AUTO)VACUUM . |
| last_automerge_timestamp | Timestamp with time zone | N/A | Last automerge time (only for HStore_opt tables). This column is supported only by 9.1.0.100 and later versions. |
| last_automerge_time_cost | Bigint | N/A | Time consumed by the last automerge (only for HStore_opt tables). This column is supported only by 9.1.0.100 and later versions. |
| last_automerge_count | Bigint | N/A | Number of records in the last automerge (only for HStore_opt tables). This column is supported only by 9.1.0.100 and later versions. |
| extra1 | Bigint | N/A | Reserved column 1. |

18.2.76 PG_SUBSCRIPTION

PG_SUBSCRIPTION records all existing subscriptions.

Table 18-77 PG_SUBSCRIPTION columns

| Column | Type | Reference | Description |
|----------|------|---------------------------------|--|
| OID | OID | - | Row identifier (hidden attribute; displayed only when explicitly selected) |
| subdbid | OID | PG_DATABASE.oid | OID of the database that the subscription belongs to |
| subname | Name | - | Name of a subscription |
| subowner | OID | PG_AUTHID.oid | Owner of a subscription |

| Column | Type | Reference | Description |
|-----------------|---------|-----------|---|
| subenabled | Boolean | - | If it is true , the subscription is enabled and should be replicated. |
| subconninfo | Text | - | Information about the connection to the database at the publisher end |
| subslotname | Text | - | Name of the replication slot in the publisher database. If this parameter is left blank, the value is NONE . |
| subpublications | Text[] | - | Array of subscribed publication names. These are the references to the publications on the publisher server. |

Examples

View all subscriptions.

```
SELECT * FROM pg_subscription;
 subdbid | subname | subowner | subenabled |
 subconninfo | subslotname | subpublications
-----+-----+-----+-----+-----+-----+-----
15992 | mysub | 10 | t | host=1.1.1.1,2.2.2.2 port=10000,20000 dbname=postgres user=repusr1
password=password | mysub | {mypub}
(1 row)
```

18.2.77 PG_SYNONYM

PG_SYNONYM records the mapping between synonym object names and other database object names.

Table 18-78 PG_SYNONYM columns

| Column | Type | Description |
|--------------|------|---|
| synname | Name | Synonym name. |
| synnamespace | OID | OID of the namespace where the synonym is located. |
| synowner | OID | Owner of a synonym, usually the OID of the user who created it. |
| synobjschema | Name | Schema name specified by the associated object. |
| synobjname | Name | Name of the associated object. |

18.2.78 PG_TABLESPACE

PG_TABLESPACE records tablespace information.

Table 18-79 PG_TABLESPACE columns

| Column | Type | Description |
|------------|-----------|--|
| spcname | Name | Name of the tablespace. |
| spcowner | OID | Owner of the tablespace, usually the user who created it. |
| spcacl | aclitem[] | Access permissions. For details, see sections "GRANT" and "REVOKE" in <i>Data Warehouse Service (DWS) SQL Syntax Reference</i> . |
| spcoptions | Text[] | Options of the tablespace. |
| spcmaxsize | Text | Maximum size of the available disk space, in bytes. |

18.2.79 PG_TRIGGER

PG_TRIGGER records the trigger information.

| Column | Type | Description |
|---------------|----------|---|
| tgrelid | OID | OID of the table where the trigger is located. |
| tgname | Name | Trigger name. |
| tgfoid | OID | Trigger OID. |
| tgtype | Smallint | Trigger type. |
| tgenabled | Char | O : The trigger fires in "origin" or "local" mode.
D : The trigger is disabled.
R : The trigger fires in "replica" mode.
A : The trigger always fires. |
| tgisinternal | Boolean | Internal trigger ID. If the value is true, it indicates an internal trigger. |
| tgconstrrelid | OID | Table referenced by the integrity constraint. |
| tgconstrindid | OID | Index of the integrity constraint. |
| tgconstraint | OID | OID of the constraint trigger in pg_constraint . |
| tgdeferrable | Boolean | The constraint trigger is of the DEFERRABLE type. |

| Column | Type | Description |
|----------------|--------------|---|
| tginitdeferred | Boolean | whether the trigger is of the INITIALLY DEFERRED type. |
| tgargs | Smallint | Input parameters number of the trigger function. |
| tgattr | int2vector | Column ID specified by the trigger. If no column is specified, an empty array is used. |
| tgargs | bytea | Parameter transferred to the trigger. |
| tgqual | pg_node_tree | Indicates the WHEN condition of the trigger. If the WHEN condition does not exist, the value is null. |

18.2.80 PG_TS_CONFIG

PG_TS_CONFIG records entries representing text search configurations. A configuration specifies a particular text search parser and a list of dictionaries to use for each of the parser's output token types.

The parser is shown in the **PG_TS_CONFIG** entry, but the token-to-dictionary mapping is defined by subsidiary entries in [PG_TS_CONFIG_MAP](#).

Table 18-80 PG_TS_CONFIG columns

| Name | Type | Reference | Description |
|--------------|--------|-----------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| cfgname | Name | N/A | Text search configuration name |
| cfgnamespace | OID | PG_NAMESPACE .oid | OID of the namespace where the configuration resides |
| cfgowner | OID | PG_AUTHID .oid | Owner of the configuration |
| cfgparser | OID | PG_TS_PARSER .oid | OID of the text search parser for this configuration |
| cfgoptions | Text[] | N/A | Configuration options |

18.2.81 PG_TS_CONFIG_MAP

PG_TS_CONFIG_MAP records entries showing which text search dictionaries should be consulted, and in what order, for each output token type of each text search configuration's parser.

Table 18-81 PG_TS_CONFIG_MAP columns

| Name | Type | Reference | Description |
|--------------|---------|-----------------------------------|---|
| mapcfg | OID | PG_TS_CONFIG .oid | OID of the PG_TS_CONFIG entry owning this map entry |
| maptokentype | Integer | N/A | A token type emitted by the configuration's parser |
| mapseqno | Integer | N/A | Order in which to consult this entry |
| mapdict | OID | PG_TS_DICT .oid | OID of the text search dictionary to consult |

18.2.82 PG_TS_DICT

PG_TS_DICT records entries that define text search dictionaries. A dictionary depends on a text search template, which specifies all the implementation functions needed. The dictionary itself provides values for the user-settable parameters supported by the template.

This division of labor allows dictionaries to be created by unprivileged users. The parameters are specified by a text string **dictinitoption**, whose format and meaning vary depending on the template.

Table 18-82 PG_TS_DICT columns

| Name | Type | Reference | Description |
|----------------|------|-------------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| dictname | Name | N/A | Text search dictionary name |
| dictnamespace | OID | PG_NAMESPACE .oid | OID of the namespace that contains the dictionary |
| dictowner | OID | PG_AUTHID .oid | Owner of the dictionary |
| dicttemplate | OID | PG_TS_TEMPLATE .oid | OID of the text search template for this dictionary |
| dictinitoption | Text | N/A | Initialization option string for the template |

18.2.83 PG_TS_PARSER

PG_TS_PARSER records entries defining text search parsers. A parser splits input text into lexemes and assigns a token type to each lexeme. Since a parser must be implemented by C functions, parsers can be created only by database administrators.

Table 18-83 PG_TS_PARSER columns

| Name | Type | Reference | Description |
|-------------|---------|-----------------------------------|--|
| OID | OID | N/A | Row identifier (hidden attribute; displayed only when explicitly selected) |
| prsname | Name | N/A | Text search parser name |
| pramespace | OID | PG_NAMESPACE .oid | OID of the namespace that contains the parser |
| prsstart | regproc | PG_PROC .oid | OID of the parser's startup function |
| prstoken | regproc | PG_PROC .oid | OID of the parser's next-token function |
| prsend | regproc | PG_PROC .oid | OID of the parser's shutdown function |
| prsheadline | regproc | PG_PROC .oid | OID of the parser's headline function |
| prsllextype | regproc | PG_PROC .oid | OID of the parser's lextype function |

18.2.84 PG_TS_TEMPLATE

PG_TS_TEMPLATE records entries defining text search templates. A template provides a framework for text search dictionaries. Since a template must be implemented by C functions, templates can be created only by database administrators.

Table 18-84 PG_TS_TEMPLATE columns

| Name | Type | Reference | Description |
|----------|------|-----------|--|
| OID | OID | - | Row identifier (hidden attribute; must be explicitly selected) |
| tmplname | Name | - | Text search template name |

| Name | Type | Reference | Description |
|---------------|---------|----------------------------------|---|
| tmplnamespace | OID | PG_NAMESPACE.oid | OID of the namespace that contains the template |
| tmplinit | regproc | PG_PROC.oid | OID of the template's initialization function |
| tmpllexize | regproc | PG_PROC.oid | OID of the template's lexize function |

18.2.85 PG_TYPE

PG_TYPE records the information about data types.

Table 18-85 PG_TYPE columns

| Column | Type | Description |
|--------------|----------|--|
| typname | Name | Data type name |
| typnamespace | OID | OID of the namespace that contains this type |
| typowner | OID | Owner of this type |
| typplen | Smallint | Number of bytes in the internal representation of the type for a fixed-size type. But for a variable-length type, typplen is negative. <ul style="list-style-type: none">• -1 indicates a "varlena" type (one that has a length word).• -2 indicates a null-terminated C string. |
| typbyval | boolean | Whether the value of this type is passed by parameter or reference of this column. If the value of TYPLEN is not 1, 2, 4, or 8, you are advised to set TYPBYVAL to false , because values of this type are always passed by reference of this column. TYPBYVAL can be false even if TYPLEN allows passing values. |
| typtype | Char | <ul style="list-style-type: none">• b indicates a basic type.• c indicates a composite type, for example, a table's row type.• e indicates an enumeration type.• p indicates a pseudo type. For details, see typrelid and typbasetype . |
| typcategory | Char | typcategory is an arbitrary classification of data types that is used by the parser to determine which implicit casts should be "preferred". |

| Column | Type | Description |
|----------------|---------|---|
| typispreferred | boolean | Whether data is converted. It is true if conversion is performed when data meets the conversion rules specified by TYPCATEGORY . |
| typisdefined | boolean | The value is true if the type is defined. The value is false if this is a placeholder entry for a not-yet-defined type. When it is false , type name, namespace, and OID are the only dependable objects. |
| typdelim | Char | Character that separates two values of this type when parsing array input. Note that the delimiter is associated with the array element data type, not the array data type. |
| typrelid | OID | If this is a composite type (see typtype), then this column points to the pg_class entry that defines the corresponding table. For a free-standing composite type, the pg_class entry does not represent a table, but it is required for the type's pg_attribute entries to link to. The value is 0 for non-composite types. |
| typelem | OID | If typelem is not 0 then it identifies another row in pg_type . The current type can be subscripted like an array yielding values of type typelem . The current type can then be subscripted like an array yielding values of type typelem . A "true" array type is variable length (typlen = -1), but some fixed-length (typlen > 0) types also have nonzero typelem , for example name and point . If a fixed-length type has a typelem , its internal representation must be some number of values of the typelem data type with no other data. Variable-length array types have a header defined by the array subroutines. |
| typarray | OID | Indicates that the corresponding type record is available in pg_type if the value is not 0 . |
| typinput | regproc | Input conversion function (text format) |
| typoutput | regproc | Output conversion function (text format) |
| typreceive | regproc | Input conversion function (binary format). If no input conversion function, the value is 0 . |
| typsend | regproc | output conversion function (binary format). If no output conversion function, the value is 0 . |
| typmodin | regproc | Type modifier input function. The value is 0 if the type does not support modifiers. |
| typmodout | regproc | Type modifier output function. The value is 0 if the type does not support modifiers. |

| Column | Type | Description |
|-------------|---------|--|
| typanalyze | regproc | Custom ANALYZE function. The value is 0 if the standard function is used. |
| typalign | Char | <p>Alignment required when storing a value of this type. It applies to storage on disk as well as most representations of the value inside PostgreSQL. When multiple values are stored consecutively, such as in the representation of a complete row on disk, padding is inserted before a data of this type so that it begins on the specified boundary. The alignment reference is the beginning of the first datum in the sequence. Possible values are:</p> <ul style="list-style-type: none">• c: char alignment, that is, no alignment needed• s: short alignment (2 bytes on most machines)• i: int alignment (4 bytes on most machines).• d: double alignment (8 bytes on many machines, but by no means all) <p>NOTICE
For types used in system tables, the size and alignment defined in pg_type must agree with the way that the compiler lays out the column in a structure representing a table row.</p> |
| typstorage | Char | <p>typstorage tells for varlena types (those with typlen = -1) if the type is prepared for toasting and what the default strategy for attributes of this type should be. Possible values are:</p> <ul style="list-style-type: none">• p indicates that values are always stored plain.• e: Value can be stored in a "secondary" relationship (if the relation has one, see pg_class.reltoastrelid).• m: Values can be stored compressed inline.• x: Values can be stored compressed inline or stored in secondary storage. <p>NOTICE
m domains can also be moved out to secondary storage, but only as a last resort (e and x domains are moved first).</p> |
| typnotnull | boolean | Represents a NOTNULL constraint on a type. Currently, it is used for domains only. |
| typbasetype | OID | If this is a domain (see typtype), then typbasetype identifies the type that this one is based on. The value is 0 if this type is not a derived type. |
| typtypmod | Integer | Records the typtypmod to be applied to domains' base types by domains (the value is -1 if the base type does not use typmod). The value is -1 if this type is not a domain. |

| Column | Type | Description |
|---------------|--------------|--|
| typndims | Integer | Number of array dimensions for a domain that is an array (that is, typbasetype is an array type; the domain's typelem matches the base type's typelem). The value is 0 for types other than domains over array types. |
| typcollation | OID | Sequence rule for specified types. Sequencing is not supported if the value is 0. |
| typdefaultbin | pg_node_tree | nodeToString() representation of a default expression for the type if the value is non-null. Currently, this column is only used for domains. |
| typdefault | Text | The value is null if a type has no associated default value. If typdefaultbin is not null, typdefault must contain a human-readable version of the default expression represented by typdefaultbin . If typdefaultbin is null and typdefault is not, then typdefault is the external representation of the type's default value, which can be fed to the type's input converter to produce a constant. |
| typacl | aclitem[] | Access permissions |

18.2.86 PG_USER_MAPPING

PG_USER_MAPPING records the mappings from local users to remote.

It is accessible only to users with system administrator rights. You can use view [PG_USER_MAPPINGS](#) to query common users.

Table 18-86 PG_USER_MAPPING columns

| Column | Type | Reference | Description |
|-----------|--------|---------------------------------------|---|
| OID | OID | - | Row identifier (hidden attribute; must be explicitly selected) |
| umuser | OID | PG_AUTHID.oid | OID of the local role being mapped, 0 if the user mapping is public |
| umserver | OID | PG_FOREIGN_SERVER.oid | OID of the foreign server that contains this mapping |
| umoptions | Text[] | - | Option used for user mapping. It is a keyword=value string. |

18.2.87 PG_USER_STATUS

PG_USER_STATUS records the states of users that access to the database. It is accessible only to users with system administrator rights.

Table 18-87 PG_USER_STATUS columns

| Column | Type | Description |
|---------------|--------------------------|--|
| roloid | OID | ID of the role |
| failcount | Integer | Specifies the number of failed attempts. |
| locktime | Timestamp with time zone | Time at which the role is locked |
| rolstatus | Smallint | Role state <ul style="list-style-type: none">• 0: normal• 1 indicates that the role is locked for some time because the failed login attempts exceed the threshold• 2 indicates that the role is locked by the administrator. |
| permspac
e | Bigint | Size of the permanent table storage space used by a role in the current instance. |
| tempspac
e | Bigint | Size of the temporary table storage space used by a role in the current instance. |

18.2.88 PG_WORKLOAD_ACTION

PG_WORKLOAD_ACTION records information about **query_band**.

Table 18-88 PG_WORKLOAD_ACTION columns

| Column | Type | Description |
|--------|------|--|
| qband | Name | query_band key-value pairs |
| class | Name | Class of the object associated with query_band |
| object | Name | Object associated with query_band |
| action | Name | Action of the object associated with query_band |

18.2.89 PGXC_CLASS

PGXC_CLASS records the replicated or distributed information for each table.

Table 18-89 PGXC_CLASS columns

| Column | Type | Description |
|-----------------|------------------|--|
| pcrelid | OID | Table OID |
| plocator_type | Char | Locator type <ul style="list-style-type: none">• H: hash• M: Modulo• N: Round Robin• R: Replicate |
| pchashalgorithm | Smallint | Distributed tuple using the hash algorithm |
| pchashbuckets | Smallint | Value of a hash container |
| pgroup | Name | Node group name |
| redistributed | Char | Whether a table has been redistributed |
| redis_order | Integer | Redistribution sequence |
| pccatnum | int2vector | Column number used as a distribution key |
| nodeoids | oidvector_extend | List of distributed table node OIDs |
| options | Text | Extension status information, which is a reserved column in the system |

18.2.90 PGXC_GROUP

PGXC_GROUP records node group information. In storage-compute decoupling 3.0 version, each node group in a logical cluster is called a Virtual Warehouse (VW). At the storage KV layer, each VW corresponds to a vgroup.

Table 18-90 PGXC_GROUP columns

| Column | Type | Description |
|------------|------|-----------------|
| group_name | Name | Node group name |

| Column | Type | Description |
|---------------------|--------------------------|--|
| in_redistribution | Char | Whether redistribution is required <ul style="list-style-type: none">• n indicates that the NodeGroup is not redistributed.• y indicates the source NodeGroup in redistribution.• t indicates the destination NodeGroup in redistribution.• s indicates that the NodeGroup will skip redistribution. |
| group_members | oidvector_extend | Node OID list of the node group |
| group_buckets | Text | Distributed data bucket group |
| is_installation | boolean | Whether to install a sub-cluster |
| group_acl | aclitem[] | Access permissions |
| group_kind | Char | Node group type. <ul style="list-style-type: none">• i indicates the installation node group, which contains all DNs.• n indicates a common non-logical cluster node group.• v indicates a logical cluster node group.• e indicates the elastic cluster node group.• r indicates a replication table node group, which can only be used to create replication tables and can contain one or more logical cluster node groups. |
| group_ckpt_csn | Xid | CSN of the last incremental extraction performed on a node group |
| vgroup_id | Xid | ID of the vgroup corresponding to the node group |
| vgroup_bucket_count | OID | Number of buckets in the vgroup corresponding to the node group |
| group_ckpt_time | Timestamp with time zone | Physical time when the last incremental extraction is performed on a node group |
| apply_kv_duration | Integer | Duration of incremental scanning in the last incremental extraction of a node group, in seconds |

| Column | Type | Description |
|---------------|---------|--|
| ckpt_duration | Integer | Checkpoint duration in the last incremental extraction of a node group, in seconds |

18.2.91 PGXC_NODE

PGXC_NODE records information about cluster nodes.

Table 18-91 PGXC_NODE columns

| Column | Type | Description |
|------------------|---------|---|
| node_name | Name | Node name |
| node_type | Char | Node type
C : CN
D : DN |
| node_port | Integer | Port ID of the node |
| node_host | Name | Host name or IP address of a node. (If a virtual IP address is configured, its value is a virtual IP address.) |
| node_port1 | Integer | Port number of a replication node |
| node_host1 | Name | Host name or IP address of a replication node. (If a virtual IP address is configured, its value is a virtual IP address.) |
| hostis_primary | boolean | Whether a switchover occurs between the primary and the standby server on the current node |
| nodeis_primary | boolean | Whether the current node is preferred to execute non-query operations in the replication table |
| nodeis_preferred | boolean | Whether the current node is preferred to execute queries in the replication table |
| node_id | Integer | Node identifier |
| sctp_port | Integer | Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the data channel. |

| Column | Type | Description |
|----------------|---------|--|
| control_port | Integer | Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the control channel. |
| sctp_port1 | Integer | Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the data channel. |
| control_port1 | Integer | Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the control channel. |
| nodeis_central | boolean | Indicates that the current node is the central node. |

Examples

Query the CN and DN information of the cluster.

```
SELECT * FROM pgxc_node;
 node_name | node_type | node_port | node_host | node_port1 | node_host1 | hostis_primary |
nodeis_primary | nodeis_preferred | node_id
| sctp_port | control_port | sctp_port1 | control_port1 | nodeis_central | read_only
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
datanode1 | D      | 55504 | localhost | 55504 | localhost | t      | f      | f      |
888802358
| 55505 | 55507 | 0 | 0 | f      | f      |
datanode2 | D      | 55508 | localhost | 55508 | localhost | t      | f      | f      |
-905831925
| 55509 | 55511 | 0 | 0 | f      | f      |
coordinator1 | C      | 55500 | localhost | 55500 | localhost | t      | f      | f      |
1938253334
| 0 | 0 | 0 | 0 | t      | f      |
datanode3 | D      | 55542 | localhost | 55542 | localhost | t      | f      | f      |
-1894792127
| 57552 | 55544 | 0 | 0 | f      | t      |
datanode4 | D      | 55546 | localhost | 55546 | localhost | t      | f      | f      |
-1307323892
| 57808 | 55548 | 0 | 0 | f      | t      |
datanode5 | D      | 55550 | localhost | 55550 | localhost | t      | f      | f      |
1797586929
| 58064 | 55552 | 0 | 0 | f      | t      |
datanode6 | D      | 55554 | localhost | 55554 | localhost | t      | f      | f      |
587455710
| 58320 | 55556 | 0 | 0 | f      | t      |
datanode7 | D      | 55558 | localhost | 55558 | localhost | t      | f      | f      |
-1685037427
| 58576 | 55560 | 0 | 0 | f      | t      |
datanode8 | D      | 55562 | localhost | 55562 | localhost | t      | f      | f      |
-993847320
| 58832 | 55564 | 0 | 0 | f      | t      |
(9 rows)
```

18.2.92 PLAN_TABLE_DATA

PLAN_TABLE_DATA stores the plan information collected by **EXPLAIN PLAN**. Different from the **PLAN_TABLE** view, the system catalog **PLAN_TABLE_DATA** stores the plan information collected by all sessions and users.

Table 18-92 PLAN_TABLE columns

| Column | Type | Description |
|--------------|----------------|--|
| session_id | Text | Session that inserts the data. Its value consists of a service thread start timestamp and a service thread ID. Values are constrained by NOT NULL . |
| user_id | OID | User who inserts the data. Values are constrained by NOT NULL . |
| statement_id | varchar2(30) | Query tag specified by a user |
| plan_id | Bigint | ID of a plan to be queried |
| id | Int | Node ID in a plan |
| operation | varchar2(30) | Operation description |
| options | varchar2(255) | Operation parameters |
| object_name | Name | Name of an operated object. It is defined by users. |
| object_type | varchar2(30) | Object type |
| object_owner | Name | User-defined schema to which an object belongs |
| projection | varchar2(4000) | Returned column information |

NOTE

- **PLAN_TABLE_DATA** records data of all users and sessions on the current node. Only administrators can access all the data. Common users can view only their own data in the **PLAN_TABLE** view.
- Data of inactive (exited) sessions is cleaned from **PLAN_TABLE_DATA** by **gs_clean** after being stored in this system catalog for a certain period of time (5 minutes by default). You can also manually run **gs_clean -C** to delete inactive session data from the table..
- Data is automatically inserted into **PLAN_TABLE_DATA** after **EXPLAIN PLAN** is executed. Therefore, do not manually insert data into or update data in **PLAN_TABLE_DATA**. Otherwise, data in **PLAN_TABLE_DATA** may be disordered. To delete data from **PLAN_TABLE_DATA**, you are advised to use the **PLAN_TABLE** view.
- Information in the **statement_id**, **object_name**, **object_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.

18.2.93 SNAPSHOT

SNAPSHOT records the start and end time of each performance view snapshot creation. After **enable_wdr_snapshot** is set to **on**, this catalog is created and maintained by the background snapshot thread. It is accessible only to users with system administrator rights.

NOTICE

- This system catalog's schema is **dbms_om**.
- Do not modify or delete this catalog externally. Otherwise, functions related to view snapshots may not work properly.

Table 18-93 dbms_om.snapshot columns

| Column | Type | Description |
|-------------|--------------------------|---|
| snapshot_id | Name | Snapshot ID. This column is the primary key and distribution key. |
| start_ts | Timestamp with time zone | Snapshot start time. |
| end_ts | Timestamp with time zone | Snapshot end time. |

18.2.94 TABLES_SNAP_TIMESTAMP

TABLES_SNAP_TIMESTAMP records the start and end time of the snapshots created for each performance view. After **enable_wdr_snapshot** is set to **on**, this catalog is created and maintained by the background snapshot thread. It is accessible only to users with system administrator rights.

Table 18-94 dbms_om.tables_snap_timestamp columns

| Column | Type | Description |
|-------------|--------------------------|---|
| snapshot_id | Name | Snapshot ID. This column is the primary key and distribution key. |
| db_name | Text | Name of the database to which the view belongs. |
| tablename | Text | View name. |
| start_ts | Timestamp with time zone | Snapshot start time. |
| end_ts | Timestamp with time zone | Snapshot end time. |

NOTICE

- This system catalog's schema is **dbms_om**.
- Do not modify or delete this catalog externally. Otherwise, functions related to view snapshots may not work properly.

18.2.95 System Catalogs for Performance View Snapshot

After **enable_wdr_snapshot** is set to **on**, the background snapshot thread creates and maintains a system catalog named in the format of **SNAP_View name** to record the snapshot result of each performance view. The following system catalogs are accessible only to users with system administrator rights:

- SNAP_PGXC_OS_RUN_INFO
- SNAP_PGXC_WAIT_EVENTS
- SNAP_PGXC_INSTR_UNIQUE_SQL
- SNAP_PGXC_STAT_BAD_BLOCK
- SNAP_PGXC_STAT_BGWRITER
- SNAP_PGXC_STAT_REPLICATION
- SNAP_PGXC_REPLICATION_SLOTS
- SNAP_PGXC_SETTINGS
- SNAP_PGXC_INSTANCE_TIME
- SNAP_GLOBAL_WORKLOAD_TRANSACTION
- SNAP_PGXC_WORKLOAD_SQL_COUNT
- SNAP_PGXC_STAT_DATABASE
- SNAP_GLOBAL_STAT_DATABASE
- SNAP_PGXC_REDO_STAT
- SNAP_GLOBAL_REDO_STAT
- SNAP_PGXC_REL_IOSTAT
- SNAP_GLOBAL_REL_IOSTAT
- SNAP_PGXC_TOTAL_MEMORY_DETAIL
- SNAP_PGXC_NODE_STAT_RESET_TIME
- SNAP_PGXC_SQL_COUNT
- SNAP_GLOBAL_TABLE_STAT
- SNAP_GLOBAL_TABLE_CHANGE_STAT
- SNAP_GLOBAL_COLUMN_TABLE_IO_STAT
- SNAP_GLOBAL_ROW_TABLE_IO_STAT

Except the new **snapshot_id** column (of the bigint type), the definitions of the other columns in these system catalogs are the same as those of the corresponding views, and the distribution key of each system catalog is **snapshot_id**.

For example, **SNAP_PGXC_OS_RUN_INFO** is used to record snapshots of the **PGXC_OS_RUN_INFO** view. The **snapshot_id** column is new, and other columns are the same as those of the **PGXC_OS_RUN_INFO** view.

NOTICE

- The schema of all above system catalogs is **dbms_om**.
- Do not modify or delete these catalogs externally. Otherwise, functions related to view snapshots may not work properly.

18.3 System Views

18.3.1 ALL_ALL_TABLES

ALL_ALL_TABLES displays the tables or views accessible to the current user.

Table 18-95 ALL_ALL_TABLES columns

| Column | Type | Description |
|-----------------|------|---|
| owner | Name | Owner of the table or view |
| table_name | Name | Name of the table or view |
| tablespace_name | Name | Tablespace where the table or view is located |

18.3.2 ALL_CONSTRAINTS

ALL_CONSTRAINTS displays information about constraints accessible to the current user.

Table 18-96 ALL_CONSTRAINTS columns

| Column | Type | Description |
|-----------------|---------------------------|---|
| constraint_name | vcharacter
varying(64) | Constraint name |
| constraint_type | Text | Constraint type <ul style="list-style-type: none">• C: Check constraint• F: Foreign key constraint• P: Primary key constraint• U: Unique constraint. |
| table_name | character
varying(64) | Name of constraint-related table |
| index_owner | character
varying(64) | Owner of constraint-related index (only for the unique constraint and primary key constraint) |

| Column | Type | Description |
|------------|-----------------------|--|
| index_name | character varying(64) | Name of constraint-related index (only for the unique constraint and primary key constraint) |

18.3.3 ALL_CONS_COLUMNS

ALL_CONS_COLUMNS displays information about constraint columns accessible to the current user.

Table 18-97 ALL_CONS_COLUMNS columns

| Column | Type | Description |
|-----------------|-----------------------|-------------------------------------|
| table_name | character varying(64) | Name of constraint-related table |
| column_name | character varying(64) | Name of constraint-related column |
| constraint_name | character varying(64) | Constraint name |
| position | Smallint | Position of the column in the table |

18.3.4 ALL_COL_COMMENTS

ALL_COL_COMMENTS displays column comments of tables and views that the current user can access.

Table 18-98 ALL_COL_COMMENTS columns

| Column | Type | Description |
|-------------|-----------------------|----------------------------|
| column_name | character varying(64) | Column name |
| table_name | character varying(64) | Table or view name |
| owner | character varying(64) | Owner of the table or view |
| comments | Text | Comments |

18.3.5 ALL_DEPENDENCIES

ALL_DEPENDENCIES displays dependencies between functions and advanced packages accessible to the current user.

NOTICE

Currently in DWS, this table is empty without any record due to information constraints.

Table 18-99 ALL_DEPENDENCIES columns

| Column | Type | Description |
|----------------------|------------------------|---|
| owner | character varying(30) | Owner of the object |
| Name | character varying(30) | Object name |
| type | character varying(17) | Object type |
| referenced_owner | character varying(30) | Owner of the referenced object |
| referenced_name | character varying(64) | Name of the referenced object |
| referenced_type | character varying(17) | Type of the referenced object |
| referenced_link_name | character varying(128) | Name of the link to the referenced object |
| schemaid | Numeric | ID of the current schema |
| dependency_type | character varying(4) | Dependency type (REF or HARD) |

18.3.6 ALL_IND_COLUMNS

ALL_IND_COLUMNS displays all index columns accessible to the current user.

Table 18-100 ALL_IND_COLUMNS columns

| Column | Type | Description |
|-----------------|-----------------------|-----------------------------------|
| index_owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_owner | character varying(64) | Table owner |
| table_name | character varying(64) | Table name |
| column_name | Name | Column name |
| column_position | Smallint | Position of a column in the index |

18.3.7 ALL_IND_EXPRESSIONS

ALL_IND_EXPRESSIONS displays information about the expression indexes accessible to the current user.

Table 18-101 ALL_IND_EXPRESSIONS columns

| Column | Type | Description |
|-------------------|-----------------------|---|
| index_owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_owner | character varying(64) | Table owner |
| table_name | character varying(64) | Table name |
| column_expression | Text | Function-based index expression of a specified column |
| column_position | Smallint | Position of a column in the index |

18.3.8 ALL_INDEXES

ALL_INDEXES displays information about indexes accessible to the current user.

Table 18-102 ALL_INDEXES columns

| Column | Type | Description |
|-------------|-----------------------|---|
| owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_name | character varying(64) | Name of the table corresponding to the index |
| uniqueness | Text | Whether the index is unique |
| generated | character varying(1) | Whether the index name is generated by the system |
| partitioned | character(3) | Whether the index has the property of the partition table |

18.3.9 ALL_OBJECTS

ALL_OBJECTS displays all database objects accessible to the current user.

Table 18-103 ALL_OBJECTS columns

| Column | Type | Description |
|---------------|--------------------------|--|
| owner | Name | Owner of the object |
| object_name | Name | Object name |
| object_id | OID | OID of the object |
| object_type | Name | Type of the object |
| namespace | OID | Namespace containing the object |
| created | Timestamp with time zone | Object creation time |
| last_ddl_time | Timestamp with time zone | Last time when the object was modified |

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see [PG_OBJECT](#).

18.3.10 ALL_PROCEDURES

ALL_PROCEDURES displays information about all stored procedures or functions accessible to the current user.

Table 18-104 ALL_PROCEDURES columns

| Column | Type | Description |
|-------------|------|---------------------|
| owner | Name | Owner of the object |
| object_name | Name | Object name |

18.3.11 ALL_SEQUENCES

ALL_SEQUENCES displays all sequences accessible to the current user.

Table 18-105 ALL_SEQUENCES columns

| Column | Type | Description |
|----------------|--------------|---|
| sequence_owner | Name | Owner of the sequence |
| sequence_name | Name | Name of the sequence |
| min_value | Bigint | Minimum value of the sequence |
| max_value | Bigint | Maximum value of the sequence |
| increment_by | Bigint | Value by which the sequence is incremented |
| cycle_flag | character(1) | Whether the sequence is a cycle sequence. The value can be Y or N . <ul style="list-style-type: none">• Y: It is a cycle sequence.• N: It is not a cycle sequence. |

18.3.12 ALL_SOURCE

ALL_SOURCE displays information about stored procedures or functions accessible to the current user, and provides the columns defined by the stored procedures and functions.

Table 18-106 ALL_SOURCE columns

| Column | Type | Description |
|--------|------|--------------------------|
| owner | Name | Owner of the object |
| Name | Name | Name of the object |
| type | Name | Type of the object |
| Text | Text | Definition of the object |

18.3.13 ALL_SYNONYMS

ALL_SYNONYMS displays all synonyms accessible to the current user.

Table 18-107 ALL_SYNONYMS columns

| Column | Type | Description |
|-------------|------|---|
| owner | Text | Owner of a synonym |
| schema_name | Text | Name of the schema to which the synonym belongs |

| Column | Type | Description |
|-------------------|------|---|
| synonym_name | Text | Synonym name |
| table_owner | Text | Owner of the associated object |
| table_schema_name | Text | Name of the schema the associated object belongs to |
| table_name | Text | Name of the associated object |

18.3.14 ALL_TAB_COLUMNS

ALL_TAB_COLUMNS displays description of columns of the tables and views that the current user can access.

Table 18-108 ALL_TAB_COLUMNS columns

| Column | Type | Description |
|----------------|------------------------|---|
| owner | character varying(64) | Owner of a table/view |
| table_name | character varying(64) | Table/View name |
| column_name | character varying(64) | Column name |
| data_type | character varying(128) | Data type of a column |
| column_id | Integer | Column ID generated when an object is created or a column is added |
| data_length | Integer | Length of the column, in bytes |
| avg_col_len | Numeric | Average length of a column, in bytes |
| nullable | bpchar | Whether the column can be empty. For the primary key constraint and non-null constraint, the value is n. |
| data_precision | Integer | Precision of the data type. This parameter is valid for the numeric data type and NULL for other types. |
| data_scale | Integer | Number of decimal places. This parameter is valid for the numeric data type and 0 for other data types. |
| char_length | Numeric | Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types. |

| Column | Type | Description |
|--------|-----------------------|---|
| schema | character varying(64) | Namespace that contains the table or view. |
| kind | Text | Type of the current record. If the column belongs to a table, the value of this column is table . If the column belongs to a view, the value of this column is view . |

18.3.15 ALL_TAB_COMMENTS

ALL_TAB_COMMENTS displays comments about all tables and views accessible to the current user.

Table 18-109 ALL_TAB_COMMENTS columns

| Column | Type | Description |
|------------|-----------------------|----------------------------|
| owner | character varying(64) | Owner of the table or view |
| table_name | character varying(64) | Name of the table or view |
| comments | Text | Comments |

18.3.16 ALL_TABLES

ALL_TABLES displays all the tables accessible to the current user.

Table 18-110 ALL_TABLES columns

| Column | Type | Description |
|-----------------|-----------------------|--|
| owner | character varying(64) | Owner of the table |
| table_name | character varying(64) | Name of the table |
| tablespace_name | character varying(64) | Name of the tablespace that contains the table |
| status | character varying(8) | Whether the current record is valid |
| temporary | character(1) | Whether the table is a temporary table <ul style="list-style-type: none">• Y indicates that it is a temporary table.• N indicates that it is not a temporary table. |

| Column | Type | Description |
|----------|-------------------|--|
| dropped | character varying | Whether the current record is deleted <ul style="list-style-type: none">• YES indicates that it is deleted.• NO indicates that it is not deleted. |
| num_rows | Numeric | Estimated number of rows in the table |

18.3.17 ALL_USERS

ALL_USERS displays all users of the database visible to the current user, however, it does not describe the users.

Table 18-111 ALL_USERS columns

| Column | Type | Description |
|----------|------|-----------------|
| username | Name | Username |
| user_id | OID | OID of the user |

18.3.18 ALL_VIEWS

ALL_VIEWS displays the description about all views accessible to the current user.

Table 18-112 ALL_VIEWS columns

| Column | Type | Description |
|-------------|---------|-------------------------|
| owner | Name | Owner of the view |
| view_name | Name | View name |
| text_length | Integer | Text length of the view |
| Text | Text | Text in the view |

18.3.19 DBA_DATA_FILES

DBA_DATA_FILES displays the description of database files. It is accessible only to users with system administrator rights.

Table 18-113 DBA_DATA_FILES columns

| Column | Type | Description |
|-----------------|------------------|--|
| tablespace_name | Name | Name of the tablespace to which the file belongs |
| bytes | Double precision | Length of the file in bytes |

18.3.20 DBA_USERS

DBA_USERS displays all user names in the database. It is accessible only to users with system administrator rights.

Table 18-114 DBA_USERS columns

| Column | Type | Description |
|----------|-----------------------|-------------|
| username | character varying(64) | Username |

18.3.21 DBA_COL_COMMENTS

DBA_COL_COMMENTS displays column comments in the tables and views of a database. Only users with system administrator permissions can access this view.

| Column | Type | Description |
|-------------|-----------------------|----------------------------|
| column_name | character varying(64) | Column name |
| table_name | character varying(64) | Table or view name |
| owner | character varying(64) | Owner of the table or view |
| comments | Text | Comments |

18.3.22 DBA_CONSTRAINTS

DBA_CONSTRAINTS displays information about table constraints in database. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|-----------------|------------------------|-----------------|
| constraint_name | vcharacter varying(64) | Constraint name |

| Column | Type | Description |
|-----------------|-----------------------|---|
| constraint_type | Text | Constraint type <ul style="list-style-type: none">• C: Check constraint• F: Foreign key constraint• P: Primary key constraint• U: Unique constraint. |
| table_name | character varying(64) | Name of constraint-related table |
| index_owner | character varying(64) | Owner of constraint-related index (only for the unique constraint and primary key constraint) |
| index_name | character varying(64) | Name of constraint-related index (only for the unique constraint and primary key constraint) |

18.3.23 DBA_CONS_COLUMNS

DBA_CONS_COLUMNS displays information about constraint columns in database tables. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|-----------------|-----------------------|-------------------------------------|
| table_name | character varying(64) | Name of constraint-related table |
| column_name | character varying(64) | Name of constraint-related column |
| constraint_name | character varying(64) | Constraint name |
| position | Smallint | Position of the column in the table |

18.3.24 DBA_IND_COLUMNS

DBA_IND_COLUMNS displays column information about all indexes in the database. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|-------------|-----------------------|-------------|
| index_owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_owner | character varying(64) | Table owner |

| Column | Type | Description |
|-----------------|-----------------------|-----------------------------------|
| table_name | character varying(64) | Table name |
| column_name | Name | Column name |
| column_position | Smallint | Position of a column in the index |

18.3.25 DBA_IND_EXPRESSIONS

DBA_IND_EXPRESSIONS displays the information about expression indexes in the database. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|-------------------|-----------------------|---|
| index_owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_owner | character varying(64) | Table owner |
| table_name | character varying(64) | Table name |
| column_expression | Text | Function-based index expression of a specified column |
| column_position | Smallint | Position of a column in the index |

18.3.26 DBA_IND_PARTITIONS

DBA_IND_PARTITIONS displays information about all index partitions in the database. Each index partition of a partitioned table in the database, if present, has a row of records in **DBA_IND_PARTITIONS**. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|----------------|-----------------------|---|
| index_owner | character varying(64) | Name of the owner of the partitioned table index to which the index partition belongs |
| schema | character varying(64) | Schema of the partitioned index to which the index partition belongs |
| index_name | character varying(64) | Index name of the partitioned table to which the index partition belongs |
| partition_name | character varying(64) | Name of the index partition |

| Column | Type | Description |
|------------------------|---------|--|
| index_partition_usable | boolean | Whether the index partition is available |
| high_value | Text | Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set.

Reserved field for forward compatibility. The parameter pretty_high_value is added in version 8.1.3 to record the information. |
| pretty_high_value | Text | Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set.

The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of high_value . The output information can be collation and column data type. |
| def_tablespace_name | Name | Tablespace name of the index partition |

18.3.27 DBA_INDEXES

DBA_INDEXES displays all indexes in the database. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|------------|-----------------------|---|
| owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_name | character varying(64) | Name of the table corresponding to the index |
| uniqueness | Text | Whether the index is unique |
| generated | character varying(1) | Whether the index name is generated by the system |

| Column | Type | Description |
|-------------|--------------|---|
| partitioned | character(3) | Whether the index has the property of the partition table |

18.3.28 DBA_OBJECTS

DBA_OBJECTS displays all database objects in the database. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|---------------|--------------------------|--|
| owner | Name | Owner of the object |
| object_name | Name | Object name |
| object_id | OID | OID of the object |
| object_type | Name | Type of the object |
| namespace | OID | Namespace containing the object |
| created | Timestamp with time zone | Object creation time |
| last_ddl_time | Timestamp with time zone | Last time when the object was modified |

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see [PG_OBJECT](#).

18.3.29 DBA_PART_INDEXES

DBA_PART_INDEXES displays information about all partitioned table indexes in the database. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|-------------|-----------------------|--|
| index_owner | character varying(64) | Name of the owner of the partitioned table index |
| schema | character varying(64) | Schema of the partitioned table index |
| index_name | character varying(64) | Name of the partitioned table index |

| Column | Type | Description |
|------------------------|-----------------------|---|
| table_name | character varying(64) | Name of the partitioned table to which the partitioned table index belongs |
| partitioning_type | Text | Partition policy of the partitioned table
NOTE
Currently, only range partitioning and list partitioning are supported. |
| partition_count | Bigint | Number of index partitions of the partitioned table index |
| def_tablespace_name | Name | Tablespace name of the partitioned table index |
| partitioning_key_count | Integer | Number of partition keys of the partitioned table |

18.3.30 DBA_PART_TABLES

DBA_PART_TABLES displays information about all partitioned tables in the database. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|------------------------|-----------------------|---|
| table_owner | character varying(64) | Name of the owner of the partitioned table |
| schema | character varying(64) | Schema of the partitioned table |
| table_name | character varying(64) | Name of the partitioned table |
| partitioning_type | Text | Partition policy of the partitioned table
NOTE
Currently, only range partitioning and list partitioning are supported. |
| partition_count | Bigint | Number of partitions of the partitioned table |
| def_tablespace_name | Name | Tablespace name of the partitioned table |
| partitioning_key_count | Integer | Number of partition keys of the partitioned table |

18.3.31 DBA_PROCEEDURES

DBA_PROCEEDURES displays information about all stored procedures and functions in the database. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|-----------------|-----------------------|--|
| owner | character varying(64) | Owner of the stored procedure or the function |
| object_name | character varying(64) | Name of the stored procedure or the function |
| argument_number | Smallint | Number of the input parameters in the stored procedure |

18.3.32 DBA_SEQUENCES

DBA_SEQUENCES displays information about all sequences in the database. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|----------------|-----------------------|-----------------------|
| sequence_owner | character varying(64) | Owner of the sequence |
| sequence_name | character varying(64) | Name of the sequence |

18.3.33 DBA_SOURCE

DBA_SOURCE displays all stored procedures or functions in the database, and it provides the columns defined by the stored procedures or functions. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|--------|-----------------------|--|
| owner | character varying(64) | Owner of the stored procedure or the function |
| Name | character varying(64) | Name of the stored procedure or the function |
| Text | Text | Definition of the stored procedure or the function |

18.3.34 DBA_SYNONYMS

DBA_SYNONYMS displays all synonyms in the database. It is accessible only to users with system administrator rights.

Table 18-115 DBA_SYNONYMS columns

| Column | Type | Description |
|-------------------|------|---|
| owner | Text | Owner of a synonym |
| schema_name | Text | Name of the schema to which the synonym belongs |
| synonym_name | Text | Synonym name |
| table_owner | Text | Owner of the associated object |
| table_schema_name | Text | Name of the schema the associated object belongs to |
| table_name | Text | Name of the associated object |

18.3.35 DBA_TAB_COLUMNS

DBA_TAB_COLUMNS stores the columns of tables and views. Each column of a table in the database has a row in **DBA_TAB_COLUMNS**. Only users with system administrator permissions can access this view.

| Column | Type | Description |
|-------------|------------------------|--|
| owner | character varying(64) | Owner of a table/view |
| table_name | character varying(64) | Table/View name |
| column_name | character varying(64) | Column name |
| data_type | character varying(128) | Data type of the column |
| column_id | Integer | Sequence number of the column when a table/view is created |
| data_length | Integer | Length of the column, in bytes |
| comments | Text | Comments |
| avg_col_len | Numeric | Average length of a column, in bytes |
| nullable | bpchar | Whether the column can be empty. For the primary key constraint and non-null constraint, the value is n. |

| Column | Type | Description |
|----------------|-----------------------|---|
| data_precision | Integer | Precision of the data type. This parameter is valid for the numeric data type and NULL for other data types. |
| data_scale | Integer | Number of decimal places. This parameter is valid for the numeric data type and 0 for other data types. |
| char_length | Numeric | Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types. |
| schema | character varying(64) | Namespace that contains the table or view. |
| kind | Text | Type of the current record. If the column belongs to a table, the value of this column is table . If the column belongs to a view, the value of this column is view . |

18.3.36 DBA_TAB_COMMENTS

DBA_TAB_COMMENTS displays comments about all tables and views in the database. It is accessible only to users with system administrator rights.

| Column | Type | Description |
|------------|-----------------------|----------------------------|
| owner | character varying(64) | Owner of the table or view |
| table_name | character varying(64) | Name of the table or view |
| comments | Text | Comments |

18.3.37 DBA_TAB_PARTITIONS

DBA_TAB_PARTITIONS displays information about all partitions in the database.

| Column | Type | Description |
|----------------|-----------------------|--|
| table_owner | character varying(64) | Owner of the table that contains the partition |
| schema | character varying(64) | Schema of the partitioned table |
| table_name | character varying(64) | Table name |
| partition_name | character varying(64) | Name of the partition |

| Column | Type | Description |
|-------------------|------|---|
| high_value | Text | Upper boundary of a range partition or boundary value set of a list partition

Reserved field for forward compatibility. The parameter pretty_high_value is added in version 8.1.3 to record the information. |
| pretty_high_value | Text | Upper boundary of a range partition or boundary value set of a list partition

The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of high_value . The output information can be collation and column data type. |
| tablespace_name | Name | Name of the tablespace that contains the partition |

Example

View the partition information of a partitioned table:

```
CREATE TABLE web_returns_p1
(
  wr_returned_date_sk integer,
  wr_returned_time_sk integer,
  wr_item_sk integer NOT NULL,
  wr_refunded_customer_sk integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE (wr_returned_date_sk)
(
  PARTITION p2016 VALUES LESS THAN(20161231),
  PARTITION p2017 VALUES LESS THAN(20171231),
  PARTITION p2018 VALUES LESS THAN(20181231),
  PARTITION p2019 VALUES LESS THAN(20191231),
  PARTITION p2020 VALUES LESS THAN(maxvalue)
);

SELECT * FROM dba_tab_partitions where table_name='web_returns_p1';
table_owner | schema | table_name | partition_name | high_value | pretty_high_value | tablespace_name
-----+-----+-----+-----+-----+-----+-----
dbadmin    | public | web_returns_p1 | p2016          | 20161231 | 20161231 | DEFAULT TABLESPACE
dbadmin    | public | web_returns_p1 | p2017          | 20171231 | 20171231 | DEFAULT TABLESPACE
dbadmin    | public | web_returns_p1 | p2018          | 20181231 | 20181231 | DEFAULT TABLESPACE
dbadmin    | public | web_returns_p1 | p2019          | 20191231 | 20191231 | DEFAULT TABLESPACE
dbadmin    | public | web_returns_p1 | p2020          | MAXVALUE | MAXVALUE | DEFAULT
TABLESPACE
(5 rows)
```

18.3.38 DBA_TABLES

DBA_TABLES displays all tables in the database. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|-----------------|-----------------------|--|
| owner | character varying(64) | Table owner |
| table_name | character varying(64) | Table name |
| tablespace_name | character varying(64) | Name of the tablespace that contains the table |
| status | character varying(8) | Whether the current record is valid |
| temporary | character(1) | Whether the table is a temporary table <ul style="list-style-type: none">• Y indicates that it is a temporary table.• N indicates that it is not a temporary table. |
| dropped | character varying | Whether the current record is deleted <ul style="list-style-type: none">• YES indicates that it is deleted.• NO indicates that it is not deleted. |
| num_rows | Numeric | Estimated number of rows in the table |

18.3.39 DBA_TABLESPACES

DBA_TABLESPACES displays information about available tablespaces. It is accessible only to users with system administrator rights.

Table 18-116 DBA_TABLESPACES columns

| Column | Type | Description |
|-----------------|-----------------------|------------------------|
| tablespace_name | character varying(64) | Name of the tablespace |

18.3.40 DBA_TRIGGERS

DBA_TRIGGERS displays information about triggers in the database. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|--------------|-----------------------|---|
| trigger_name | character varying(64) | Trigger name |
| table_name | character varying(64) | Name of the table that defines the trigger |
| table_owner | character varying(64) | Owner of the table that defines the trigger |

18.3.41 DBA_VIEWS

DBA_VIEWS displays views in the database. This view is accessible only to users with system administrator rights.

| Column | Type | Description |
|-----------|-----------------------|-------------------|
| owner | character varying(64) | Owner of the view |
| view_name | character varying(64) | View name |

18.3.42 DUAL

DUAL is automatically created by the database based on the data dictionary. It has only one text column in only one row for storing expression calculation results. It is accessible to all users.

Table 18-117 DUAL columns

| Column | Type | Description |
|--------|------|-------------------------------|
| dummy | Text | Expression calculation result |

18.3.43 GET_ALL_TSC_INFO

Obtains the TSC information of all nodes again. This view is supported only by clusters of version 8.2.1 or later.

Table 18-118 show_tsc_info() return columns

| Column | Type | Description |
|-----------|--------|---------------------------|
| node_name | text | Node name |
| tsc_mult | bigint | TSC conversion multiplier |
| tsc_shift | bigint | TSC conversion shifts |

| Column | Type | Description |
|-----------------------|---------|---|
| tsc_frequency | float8 | TSC frequency |
| tsc_use_frequency | boolean | Indicates whether to use the TSC frequency for time conversion. |
| tsc_ready | boolean | Indicates whether the TSC frequency can be used for time conversion |
| tsc_scalar_error_info | text | Error information about obtaining TSC conversion information |
| tsc_freq_error_info | text | Error information about obtaining TSC frequency information |

18.3.44 GET_TSC_INFO

Obtains the TSC information of the current node again. This view is supported only by clusters of version 8.2.1 or later.

Table 18-119 show_tsc_info() return columns

| Column | Type | Description |
|-----------------------|---------|---|
| node_name | text | Node name |
| tsc_mult | bigint | TSC conversion multiplier |
| tsc_shift | bigint | TSC conversion shifts |
| tsc_frequency | float8 | TSC frequency |
| tsc_use_frequency | boolean | Indicates whether to use the TSC frequency for time conversion. |
| tsc_ready | boolean | Indicates whether the TSC frequency can be used for time conversion |
| tsc_scalar_error_info | text | Error information about obtaining TSC conversion information |
| tsc_freq_error_info | text | Error information about obtaining TSC frequency information |

18.3.45 GLOBAL_COLUMN_TABLE_IO_STAT

GLOBAL_COLUMN_TABLE_IO_STAT provides I/O statistics of all column-store tables in the current database. The names, types, and sequences of the columns in the view are the same as those in the **GS_COLUMN_TABLE_IO_STAT** view. For details about the columns, see [Table 18-120](#). The value of each statistical column is the sum of the values of the corresponding columns of all nodes.

Table 18-120 GS_COLUMN_TABLE_IO_STAT columns

| Column | Type | Description |
|------------|--------|--|
| schemaname | Name | Namespace of a table |
| relname | Name | Table name |
| heap_read | Bigint | Number of blocks logically read in the heap |
| heap_hit | Bigint | Number of block hits in the heap |
| idx_read | Bigint | Number of blocks logically read in the index |
| idx_hit | Bigint | Number of block hits in the index |
| cu_read | Bigint | Number of logical reads in the Compression Unit |
| cu_hit | Bigint | Number of hits in the Compression Unit |
| cidx_read | Bigint | Number of indexes logically read in the Compression Unit |
| cidx_hit | Bigint | Number of index hits in the Compression Unit |

18.3.46 GLOBAL_REDO_STAT

GLOBAL_REDO_STAT displays the total statistics of XLOG redo operations on all nodes in a cluster. Except the **avgiotim** column (indicating the average redo write time of all nodes), the names of the other columns in this view are the same as those in the **PV_REDO_STAT** view. The respective meanings of the other columns are the sum of the values of the same columns in the **PV_REDO_STAT** view on each node.

Table 18-121 GLOBAL_REDO_STAT columns

| Column | Type | Description |
|-----------|--------|--|
| phywrts | Bigint | Total number of physical writes on all nodes |
| phyblkwrt | Bigint | Total number of physical write blocks on all nodes |
| wrietim | Bigint | Total physical write time of all nodes |
| avgiotim | Bigint | Average redo write time of all nodes |
| lstiotim | Bigint | Sum of the last write time of all nodes |
| miniotim | Bigint | Sum of the minimum write time of all nodes |
| maxiowtm | Bigint | Sum of the maximum write time of all nodes |

 NOTE

This view is accessible only to users with system administrator rights.

18.3.47 GLOBAL_REL_IOSTAT

GLOBAL_REL_IOSTAT displays the total disk I/O statistics of all nodes in a cluster. The name of each column in this view is the same as that in the **GS_REL_IOSTAT** view, but the column meaning is the sum of the value of the same column in the **GS_REL_IOSTAT** view on each node.

Table 18-122 GLOBAL_REL_IOSTAT columns

| Column | Type | Description |
|-----------|--------|---|
| phyrds | Bigint | Total number of disk read times of all nodes |
| phywrts | Bigint | Total number of disk write times of all nodes |
| phyblkrd | Bigint | Total number of disk pages read by all nodes |
| phyblkwrt | Bigint | Total number of disk pages written by all nodes |

 NOTE

This view is accessible only to users with system administrator rights.

18.3.48 GLOBAL_ROW_TABLE_IO_STAT

GLOBAL_ROW_TABLE_IO_STAT provides I/O statistics of all row-store tables in the current database. The names, types, and sequences of the columns in the view are the same as those in the **GS_ROW_TABLE_IO_STAT** view. For details about the columns, see [Table 18-123](#). The value of each statistical column is the sum of the values of the corresponding columns of all nodes.

Table 18-123 GS_ROW_TABLE_IO_STAT columns

| Column | Type | Description |
|------------|--------|---|
| schemaname | Name | Namespace of a table |
| relname | Name | Name of a table |
| heap_read | Bigint | Number of blocks logically read in the heap |
| heap_hit | Bigint | Number of block hits in the heap |
| idx_read | Bigint | Number of blocks logically read in the index |
| idx_hit | Bigint | Number of block hits in the index |
| toast_read | Bigint | Number of blocks logically read in the TOAST table |

| Column | Type | Description |
|-----------|--------|--|
| toast_hit | Bigint | Number of block hits in the TOAST table |
| tidx_read | Bigint | Number of indexes logically read in the TOAST table |
| tidx_hit | Bigint | Number of index hits in the TOAST table |

18.3.49 GLOBAL_STAT_DATABASE

GLOBAL_STAT_DATABASE displays the status and statistics of databases on all nodes in a cluster.

- When you query the **GLOBAL_STAT_DATABASE** view on a CN, the respective values of all columns returned, except **stats_reset** (indicating the status reset time on the current CN), are the sum of values on related nodes in the cluster. Note that the sum range varies depending on the logical meaning of each column in the **GLOBAL_STAT_DATABASE** view.
- When you query the **GLOBAL_STAT_DATABASE** view on a DN, the query result is the same as that in [Table 18-124](#).

Table 18-124 GLOBAL_STAT_DATABASE columns

| Column | Type | Description | Sum Range |
|---------------|---------|---|-----------|
| datid | OID | Database OID | - |
| datname | Name | Database name | - |
| numbackends | Integer | Number of backends currently connected to this database on the current node. This is the only column in this view that reflects the current state value. All columns return the accumulated value since the last reset. | CN |
| xact_commit | Bigint | Number of transactions in this database that have been committed on the current node | CN |
| xact_rollback | Bigint | Number of transactions in this database that have been rolled back on the current node | CN |
| blks_read | Bigint | Number of disk blocks read in this database on the current node | DN |

| Column | Type | Description | Sum Range |
|--------------|--------|---|-----------|
| blks_hit | Bigint | Number of disk blocks found in the buffer cache on the current node, that is, the number of blocks hit in the cache. (This only includes hits in the DWS buffer cache, not in the file system cache.) | DN |
| tup_returned | Bigint | Number of rows returned by queries in this database on the current node | DN |
| tup_fetched | Bigint | Number of rows fetched by queries in this database on the current node | DN |
| tup_inserted | Bigint | Number of rows inserted in this database on the current node | DN |
| tup_updated | Bigint | Number of rows updated in this database on the current node | DN |
| tup_deleted | Bigint | Number of rows deleted from this database on the current node | DN |
| conflicts | Bigint | Number of queries canceled due to database recovery conflicts on the current node (conflicts occurring only on the standby server). For details, see PG_STAT_DATABASE_CONFLICTS . | CN and DN |
| temp_files | Bigint | Number of temporary files created by this database on the current node. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the log_temp_files setting. | DN |
| temp_bytes | Bigint | Size of temporary files written to this database on the current node. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting. | DN |
| deadlocks | Bigint | Number of deadlocks in this database on the current node | CN and DN |

| Column | Type | Description | Sum Range |
|----------------|--------------------------|--|-----------|
| blk_read_time | Double precision | Time spent reading data file blocks by backends in this database on the current node, in milliseconds | DN |
| blk_write_time | Double precision | Time spent writing into data file blocks by backends in this database on the current node, in milliseconds | DN |
| stats_reset | Timestamp with time zone | Time when the database statistics are reset on the current node | - |

18.3.50 GLOBAL_TABLE_CHANGE_STAT

GLOBAL_TABLE_CHANGE_STAT displays the changes of all tables (excluding foreign tables) in the current database. The value of each column that indicates the number of times is the accumulated value since the instance was started.

Table 18-125 GLOBAL_TABLE_CHANGE_STAT columns

| Column | Type | Description |
|------------------|--------------------------|--|
| schemaname | Name | Namespace of a table |
| relname | Name | Table name |
| last_vacuum | Timestamp with time zone | Time when the last VACUUM operation is performed manually |
| vacuum_count | Bigint | Number of times of manually performing the VACUUM operation. The value is the sum of the number of times on each CN. |
| last_autovacuum | Timestamp with time zone | Time when the last VACUUM operation is performed automatically |
| autovacuum_count | Bigint | Number of times of automatically performing the VACUUM operation. The value is the sum of the number of times on each CN. |
| last_analyze | Timestamp with time zone | Time when the last non-automatic rolling analysis is performed. |

| Column | Type | Description |
|-------------------|--------------------------|---|
| analyze_count | Bigint | Number of times of performing the ANALYZE operation (both manually and automatically). The ANALYZE operation is performed on all CNs at the same time. Therefore, the value of this column is the maximum value on all CNs. |
| last_autoanalyze | Timestamp with time zone | Time when the last automatic rolling analysis is performed. |
| autoanalyze_count | Bigint | Number of times of automatically performing the ANALYZE operation. The value is the sum of the number of times on each CN. |
| last_change | Bigint | Time when the last modification (INSERT , UPDATE , or DELETE) is performed |

18.3.51 GLOBAL_TABLE_STAT

GLOBAL_TABLE_STAT displays statistics about all tables (excluding foreign tables) in the current database. The values of **live_tuples** and **dead_tuples** are real-time values, and the values of other statistical columns are accumulated values since the instance was started.

Table 18-126 GLOBAL_TABLE_STAT columns

| Column | Type | Description |
|------------------|--------|--|
| schemaname | Name | Namespace of a table |
| relname | Name | Table name |
| distribute_mode | Char | Distribution mode of a table. The meaning of this column is the same as that of the plocator_type column in the pgxc_class system catalog. |
| seq_scan | Bigint | Number of sequential scans. For a partitioned table, the sum of the number of scans of each partition is displayed. |
| seq_tuple_read | Bigint | Number of rows scanned in sequence. |
| index_scan | Bigint | Number of index scans. |
| index_tuple_read | Bigint | Number of rows scanned by the index. |

| Column | Type | Description |
|-------------------|--------|--|
| tuple_inserted | Bigint | Number of rows inserted. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed. |
| tuple_updated | Bigint | Number of rows updated. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed. |
| tuple_deleted | Bigint | Number of rows deleted. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed. |
| tuple_hot_updated | Bigint | Number of rows with HOT updates. For a replication table, the maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed. |
| live_tuples | Bigint | Number of live tuples. The maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed.
This indicator applies only to row-store tables. |
| dead_tuples | Bigint | Number of dead tuples. The maximum value of each node is displayed. For a distribution table, the sum of all nodes is displayed.
This indicator applies only to row-store tables. |

18.3.52 GLOBAL_WORKLOAD_SQL_COUNT

GLOBAL_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in all workload Cgroups in a cluster, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements.

Table 18-127 GLOBAL_WORKLOAD_SQL_COUNT columns

| Column | Type | Description |
|--------------|--------|------------------------------------|
| workload | Name | Workload Cgroup name |
| select_count | Bigint | Number of SELECT statements |
| update_count | Bigint | Number of UPDATE statements |

| Column | Type | Description |
|--------------|--------|------------------------------------|
| insert_count | Bigint | Number of INSERT statements |
| delete_count | Bigint | Number of DELETE statements |
| ddl_count | Bigint | Number of DDL statements |
| dml_count | Bigint | Number of DML statements |
| dcl_count | Bigint | Number of DCL statements |

18.3.53 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME

GLOBAL_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in all workload Cgroups in a cluster, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond.

Table 18-128 GLOBAL_WORKLOAD_SQL_ELAPSE_TIME columns

| Column | Type | Description |
|---------------------|--------|---|
| workload | Name | Workload Cgroup name |
| total_select_elapse | Bigint | Total response time of SELECT statements |
| max_select_elapse | Bigint | Maximum response time of SELECT statements |
| min_select_elapse | Bigint | Minimum response time of SELECT statements |
| avg_select_elapse | Bigint | Average response time of SELECT statements |
| total_update_elapse | Bigint | Total response time of UPDATE statements |
| max_update_elapse | Bigint | Maximum response time of UPDATE statements |
| min_update_elapse | Bigint | Minimum response time of UPDATE statements |
| avg_update_elapse | Bigint | Average response time of UPDATE statements |

| Column | Type | Description |
|---------------------|--------|---|
| total_insert_elapse | Bigint | Total response time of INSERT statements |
| max_insert_elapse | Bigint | Maximum response time of INSERT statements |
| min_insert_elapse | Bigint | Minimum response time of INSERT statements |
| avg_insert_elapse | Bigint | Average response time of INSERT statements |
| total_delete_elapse | Bigint | Total response time of DELETE statements |
| max_delete_elapse | Bigint | Maximum response time of DELETE statements |
| min_delete_elapse | Bigint | Minimum response time of DELETE statements |
| avg_delete_elapse | Bigint | Average response time of DELETE statements |

18.3.54 GLOBAL_WORKLOAD_TRANSACTION

GLOBAL_WORKLOAD_TRANSACTION provides the total transaction information about workload Cgroups on all CNs in the cluster. This view is accessible only to users with system administrator rights. It is valid only when the real-time resource monitoring function is enabled, that is, **enable_resource_track** is **on**.

Table 18-129 GLOBAL_WORKLOAD_TRANSACTION columns

| Column | Type | Description |
|------------------|--------|---|
| workload | Name | Workload Cgroup name |
| commit_counter | Bigint | Total number of submission times on each CN |
| rollback_counter | Bigint | Total number of rollback times on each CN |
| resp_min | Bigint | Minimum response time of the cluster |
| resp_max | Bigint | Maximum response time of the cluster |
| resp_avg | Bigint | Average response time on each CN |
| resp_total | Bigint | Total response time on each CN |

18.3.55 GS_ALL_CONTROL_GROUP_INFO

GS_ALL_CONTROL_GROUP_INFO displays all Cgroup information in a database.

Table 18-130 GS_ALL_CONTROL_GROUP_INFO columns

| Column | Type | Description |
|----------|--------|--|
| Name | Text | Name of the Cgroup |
| type | Text | Type of the Cgroup |
| gid | Bigint | Cgroup ID |
| classgid | Bigint | ID of the Class Cgroup to which a Workload belongs |
| class | Text | Class Cgroup |
| workload | Text | Workload Cgroup |
| shares | Bigint | CPU quota allocated to a Cgroup |
| limits | Bigint | Limit of CPUs allocated to a Cgroup |
| wdlevel | Bigint | Workload Cgroup level |
| cpucores | Text | Usage of CPU cores in a Cgroup |

18.3.56 GS_BLOCKLIST_QUERY

GS_BLOCKLIST_QUERY is used to query job blocklist and exception information. This view is obtained by associating system catalogs **GS_BLOCKLIST_QUERY** and **GS_WLM_SESSION_INFO**, and deduplicating query results. If the **GS_WLM_SESSION_INFO** table is large, the query may take a long time.

NOTICE

- The schema of the **GS_BLOCKLIST_QUERY** view is **pg_catalog**.
- The **GS_BLOCKLIST_QUERY** view can be queried only in the **postgres** database. If it is queried in other databases, an error is reported.
- Generally, constant values are ignored during unique SQL ID calculation in DML statements. However, constant values cannot be ignored in DDL, DCL, and parameter setting statements. A **unique_sql_id** may correspond to one or more queries.

Table 18-131 GS_BLOCKLIST_QUERY columns

| Column | Type | Reference | Description |
|---------------|-----------|-----------|--|
| unique_sql_id | Bigint | N/A | Query ID generated based on the query parsing tree. |
| block_list | Boolean | N/A | Check whether a job is in the blocklist. |
| except_num | Integer | N/A | Query the number of job exceptions. |
| except_time | Timestamp | N/A | Query the time when the last job exception occurred. |
| query | Text | N/A | Statement to be executed. |

18.3.57 GS_BLOCKLIST_SQL

GS_BLOCKLIST_SQL is used to query job blocklist and exception information. This view is obtained by associating system catalogs [GS_BLOCKLIST_SQL](#) and [GS_WLM_SESSION_INFO](#), and deduplicating query results. If the **GS_WLM_SESSION_INFO** table is large, the query may take a long time.

This view is supported only by 9.1.0.200 and later cluster versions.

NOTICE

- The schema stored in the **GS_BLOCKLIST_SQL** view is **pg_catalog**.
- The **GS_BLOCKLIST_SQL** view can be queried only in the **postgres** database. If it is queried in other databases, an error is reported.
- Generally, constant values are ignored during **sql_hash** calculation in DML statements. However, constant values cannot be ignored in DDL, DCL, and parameter setting statements. A **sql_hash** may correspond to one or more queries.

Table 18-132 GS_BLOCKLIST_QUERY columns

| Column | Type | Reference | Description |
|------------|---------|-----------|--|
| sql_hash | Text | N/A | sql_hash generated based on the query parsing tree. |
| block_list | Boolean | N/A | Whether a job is in the blocklist. |
| except_num | Integer | N/A | Number of job exceptions. |

| Column | Type | Reference | Description |
|-------------|-----------|-----------|--|
| except_time | Timestamp | N/A | Time when the last job exception occurred. |
| query | Text | N/A | Statement to be executed. |

18.3.58 GS_CLUSTER_RESOURCE_INFO

GS_CLUSTER_RESOURCE_INFO displays a DN resource summary.

Table 18-133 GS_CLUSTER_RESOURCE_INFO columns

| Column | Type | Description |
|---------------|---------|-------------------------------|
| min_mem_util | Integer | Minimum memory usage of a DN |
| max_mem_util | Integer | Maximum memory usage of a DN |
| min_cpu_util | Integer | Minimum CPU usage of a DN |
| max_cpu_util | Integer | Maximum CPU usage of a DN |
| min_io_util | Integer | Minimum I/O usage of a DN |
| max_io_util | Integer | Maximum I/O usage of a DN |
| used_mem_rate | Integer | Maximum physical memory usage |

18.3.59 GS_COLUMN_TABLE_IO_STAT

GS_COLUMN_TABLE_IO_STAT displays the I/O of all column-store tables of the database on the current node. The value of each statistical column is the accumulated value since the instance was started.

Table 18-134 GS_COLUMN_TABLE_IO_STAT columns

| Column | Type | Description |
|------------|--------|--|
| schemaname | Name | Namespace of a table |
| relname | Name | Table name |
| heap_read | Bigint | Number of blocks logically read in the heap |
| heap_hit | Bigint | Number of block hits in the heap |
| idx_read | Bigint | Number of blocks logically read in the index |
| idx_hit | Bigint | Number of block hits in the index |

| Column | Type | Description |
|-----------|--------|--|
| cu_read | Bigint | Number of logical reads in the Compression Unit |
| cu_hit | Bigint | Number of hits in the Compression Unit |
| cidx_read | Bigint | Number of indexes logically read in the Compression Unit |
| cidx_hit | Bigint | Number of index hits in the Compression Unit |

18.3.60 GS_OBS_READ_TRAFFIC

Collects statistics on the OBS read traffic and average read bandwidth. The statistical results are aggregated every 10 minutes. This view is supported only by clusters of version 8.2.0 or later.

| Column | Type | Description |
|--------------------|--------------------------|---|
| nodename | TEXT | Cluster node |
| hostname | TEXT | Server node |
| traffic_mb | float8 | OBS read traffic statistics during the 10 minutes before logtime |
| bandwidth_mb_per_s | float8 | Average bandwidth, in MB/s |
| reqcount | Bigint | Number of OBS reads during the 10 minutes before logtime |
| logtime | Timestamp with time zone | Time when statistics are recorded |

Examples

Query statistics on the OBS read traffic and average read bandwidth. The statistical results are aggregated every 10 minutes.

```
select * from gs_obs_read_traffic;
nodename | hostname | traffic_mb | bandwidth_mb_per_s | reqcount | logtime
-----+-----+-----+-----+-----+-----
dn_1 | rhel_10_90_45_56 | 101.959338188171 | 5.14830159670447 | 23 | 2022-11-26 09:50:00+08
(1 row)
```

18.3.61 GS_OBS_WRITE_TRAFFIC

Collects statistics on the OBS write traffic and average write bandwidth. The statistical results are aggregated every 10 minutes. This view is supported only by clusters of version 8.2.0 or later.

| Column | Type | Description |
|--------------------|--------------------------|--|
| nodename | TEXT | Cluster node |
| hostname | TEXT | Server node |
| traffic_mb | float8 | OBS write traffic statistics during the 10 minutes before logtime |
| bandwidth_mb_per_s | float8 | Average bandwidth, in MB/s |
| reqcount | Bigint | Number of OBS writes during the 10 minutes before logtime |
| logtime | Timestamp with time zone | Time when statistics are recorded |

Examples

Query statistics on the OBS write traffic and average write bandwidth. The statistical results are aggregated every 10 minutes.

```
select * from gs_obs_write_traffic;
  nodename | hostname | traffic_mb | bandwidth_mb_per_s | reqcount | logtime
-----+-----+-----+-----+-----+-----
dn_1      | rhel_10_90_45_56 | .000738143920898438 | .000289970820362525 | 12 | 2022-10-24 16:10:00+08
dn_1      | rhel_10_90_45_56 | .000354766845703125 | .000386063466694153 | 7 | 2022-10-24 18:50:00+08
dn_1      | rhel_10_90_45_56 | 9.34600830078125e-05 | .000143659648687162 | 2 | 2022-11-07 09:20:00+08
dn_1      | rhel_10_90_45_56 | 4.10079956054688e-05 | .000186667253592502 | 1 | 2022-11-07 09:30:00+08
dn_1      | rhel_10_90_45_56 | 2048.17834663391 | 27.2766632219637 | 2 | 2022-11-22 16:10:00+08
dn_1      | rhel_10_90_45_56 | 3747.23722648621 | 28.0842938534546 | 4 | 2022-11-22 16:20:00+08
(6 row)
```

18.3.62 GS_INSTR_UNIQUE_SQL

Unique SQL Definition

The database parses each received SQL text string and generates an internal parsing tree. The database traverses the parsing tree and ignores constant values in the parsing tree. In this case, an integer value is calculated using a certain algorithm. This integer is used as the Unique SQL ID to uniquely identify this type of SQL. SQL statements with the same Unique SQL ID are called Unique SQL statements.

Examples

Assume that the user enters the following SQL statements in sequence:

```
select * from t1 where id = 1;
select * from t1 where id = 2;
```

The statistics of the two SQL statements are aggregated to the same Unique SQL statement.

```
select * from t1 where id = ?;
```

GS_INSTR_UNIQUE_SQL View

The **GS_INSTR_UNIQUE_SQL** view displays the execution information about the Unique SQL statements collected by the current node, including:

- Unique SQL ID and normalized SQL text string. The normalized SQL text is described in [Examples](#). Generally, constant values are ignored during Unique SQL ID calculation in DML statements. However, constant values cannot be ignored in DDL, DCL, and parameter setting statements.
- Number of execution times (number of successful execution times) and response time (SQL execution time in the database, including the maximum, minimum, and total time)
- Cache or I/O information, including the number of physical reads and logical reads of blocks. Only information about successfully executed SQL statements on each DN is collected. The statistical value is related to factors such as the amount of data processed during query execution, used memory, whether the query is executed for multiple times, memory management policy, and whether there are other concurrent queries. The statistical value reflects the number of physical reads and logical reads of the buffer block in the entire query execution process. The statistical value may vary according to the execution time.
- Row activities, such as the number of returned rows, updated rows, inserted rows, deleted rows, sequentially scanned rows, and randomly scanned rows in the result set of the **SELECT** statement. Except that the number of rows returned by the result set is the same as the number of rows in the result set of the **SELECT** statement and is recorded only on the CN, the activity information of other rows is recorded on the DN. The statistical value reflects the row activities during the entire query execution process, including scanning and modifying related system tables, metadata tables, and data tables. The value of this parameter is related to the data volume and related parameter settings. That is, the statistical value is greater than or equal to the scanning and modification times of actual data tables.
- Time distribution, including DB_TIME/CPU_TIME/EXECUTION_TIME/ PARSE_TIME/PLAN_TIME/REWRITE_TIME/PL_EXECUTION_TIME/ PL_COMPILATION_TIME/NET_SEND_TIME/DATA_IO_TIME. For details, see [Table 18-135](#). The information is collected on both CNs and DNs and is displayed during view query.
- Number of soft and hard parsing times, such as the number of soft parsing times (cache plan) and hard parsing times (generation plan). If the cache plan is executed this time, the number of soft parsing times increases by 1. If the generation plan is regenerated this time, the number of hard parsing times increases by 1. This number is counted on both CNs and DNs and is displayed during view query.

The Unique SQL statistics function has the following restrictions:

- Detailed statistics are displayed only for successfully executed SQL statements. Otherwise, only query, node, and user information are recorded.

- If the Unique SQL statistics collection function is enabled, the CN collects statistics on all received queries, including tool and user queries.
- If an SQL statement contains multiple SQL statements or similar stored procedures, a Unique SQL statement is generated for the outermost SQL statement. The statistics of all sub-SQL statements are summarized to the Unique SQL record.
- The response time statistics of Unique SQL does not include the time of the **NET_SEND_TIME** phase. Therefore, there is no comparison between **EXECUTION_TIME** and **elapse_time**.
- **parse_time** of clauses cannot be calculated for **begin;...;commit** and similar transaction blocks.

When a common user accesses the **GS_INSTR_UNIQUE_SQL** view, only the Unique SQL information about the user is displayed. When an administrator accesses the **GS_INSTR_UNIQUE_SQL** view, all Unique SQL information about the current node is displayed. The **GS_INSTR_UNIQUE_SQL** view can be queried on both CNs and DNs. The DN displays the Unique SQL statistics of the local node, and the CN displays the complete Unique SQL statistics of the local node. That is, the CN collects the Unique SQL execution information of the CN from other CNs and DNs and displays the information. You can query the **GS_INSTR_UNIQUE_SQL** view to locate the Top SQL statements that consume different resources, providing a basis for cluster tuning and maintenance.

The background thread checks all Unique SQL statements every hour and deletes the Unique SQL statements whose **last_time** is **instr_unique_sql_timeout** hours ago.

Table 18-135 GS_INSTR_UNIQUE_SQL columns

| Column | Type | Description |
|-----------------|---------|---|
| node_name | Name | Name of the CN that receives SQL statements |
| node_id | Integer | Node ID, which is the same as the value of node_id in the pgxc_node table |
| user_name | Name | Username |
| user_id | OID | User ID |
| unique_sql_id | Bigint | Normalized Unique SQL ID |
| query | Text | Normalized SQL text |
| n_calls | Bigint | Number of successful execution times |
| min_elapse_time | Bigint | Minimum running time of the SQL statement in the database (unit: μ s) |

| Column | Type | Description |
|-------------------|--------|---|
| max_elapse_time | Bigint | Maximum running time of SQL statements in the database (unit: μ s) |
| total_elapse_time | Bigint | Total running time of SQL statements in the database (unit: μ s) |
| n_returned_rows | Bigint | Row activity - Number of rows in the result set returned by the SELECT statement |
| n_tuples_fetched | Bigint | Row activity - Randomly scan rows (column-store tables/foreign tables are not counted.) |
| n_tuples_returned | Bigint | Row activity - Sequential scan rows (Column-store tables/foreign tables are not counted.) |
| n_tuples_inserted | Bigint | Row activity - Inserted rows |
| n_tuples_updated | Bigint | Row activity - Updated rows |
| n_tuples_deleted | Bigint | Row activity - Deleted rows |
| n_blocks_fetched | Bigint | Block access times of the buffer, that is, physical read/I/O |
| n_blocks_hit | Bigint | Block hits of the buffer, that is, logical read/cache |
| n_soft_parse | Bigint | Number of soft parsing times (cache plan) |
| n_hard_parse | Bigint | Number of hard parsing times (generation plan) |

| Column | Type | Description |
|---------------------|--------|--|
| db_time | Bigint | Valid DB execution time, including the waiting time and network sending time. If multiple threads are involved in query execution, the value of DB_TIME is the sum of DB_TIME of multiple threads (unit: μ s). |
| cpu_time | Bigint | CPU execution time, excluding the sleep time (unit: μ s) |
| execution_time | Bigint | SQL execution time in the query executor, DDL statements, and statements (such as Copy statements) that are not executed by the executor are not counted (unit: μ s). |
| parse_time | Bigint | SQL parsing time (unit: μ s) |
| plan_time | Bigint | SQL generation plan time (unit: μ s) |
| rewrite_time | Bigint | SQL rewriting time (unit: μ s) |
| pl_execution_time | Bigint | Execution time of the plpgsql procedural language function (unit: μ s) |
| pl_compilation_time | Bigint | Compilation time of the plpgsql procedural language function (unit: μ s) |
| net_send_time | Bigint | Network time, including the time spent by the CN in sending data to the client and the time spent by the DN in sending data to the CN (unit: μ s) |
| data_io_time | Bigint | File I/O time (unit: μ s) |

| Column | Type | Description |
|------------|--------------------------|---|
| first_time | Timestamp with time zone | Time of the first SQL statement execution |
| last_time | Timestamp with time zone | Time of the last SQL statement execution |

18.3.63 GS_NODE_STAT_RESET_TIME

GS_NODE_STAT_RESET_TIME provides the statistics reset time of the current node and returns a timestamp with the time zone.

For details, see the **get_node_stat_reset_time()** function in "Functions and Operators" > "System Administration Functions" > "Other Functions" in the *Data Warehouse Service (DWS) SQL Syntax Reference*.

NOTE

When an instance is running, its statistics keep rising. In the following cases, the statistical values in the memory will be reset to 0:

- The instance is restarted or a cluster switchover occurs.
- The database is dropped.
- A reset operation is performed. For example, the statistics counter in the database is reset using the **pgstat_recv_resetcounter** function or the Unique SQL statements are cleared using the **reset_instr_unique_sql** function.

If any of the preceding events occurs, DWS will record the time when the statistics are reset. You can query the time using the **get_node_stat_reset_time** function.

18.3.64 GS_OBS_LATENCY

GS_OBS_LATENCY records the average latency of OBS during the 10 minutes before **logtime**. The latency is estimated based on OBS operations. This view is supported only by clusters of version 8.2.0 or later.

Table 18-136 GS_OBS_LATENCY columns

| Column | Type | Description |
|------------|--------------------------|--|
| nodename | Text | Node |
| hostname | Text | Server node. |
| latency_ms | Double precision | Average delay of OBS during the 10 minutes before logtime . The unit is ms. |
| reqcount | Bigint | Number of OBS requests during the 10 minutes before logtime . |
| logtime | Timestamp with time zone | Time when the delay information is recorded. |

18.3.65 GS_QUERY_MONITOR

Displays the running/queuing information and resource usage of ongoing queries. Only queuing and running jobs are displayed. This view can be queried only on CNs and displays only the monitoring information about the main statement. This view is supported only by clusters of 8.2.1.100 and later versions.

Table 18-137 GS_QUERY_MONITOR columns

| Column | Type | Description |
|-------------|-----------|--|
| username | Name | Name of the user who performs the query. |
| nodename | Name | Name of the CN that executes the query. |
| nodegroup | Name | Name of the cluster where the query is performed. The default cluster name is installation . |
| rpname | Name | Name of the resource pool associated with the query. |
| priority | Name | Priority of the query, which can be Rush , High , Medium , and Low . |
| xact_start | Timestamp | Start time of the transaction to which the query belongs. |
| query_start | Timestamp | Start time of query execution. |
| block_time | Bigint | Accumulated queuing time of jobs. Stored procedures and multi-statement task may be queued for multiple times. Unit: second. |
| duration | Bigint | Running time of a job, excluding the queuing time. Unit: second. |
| query_band | Text | Job ID, which can be set using the GUC parameter query_band . By default, this parameter is left blank. |
| attribute | Text | Job attributes: <ul style="list-style-type: none">● Simple: simple job.● Complicated: complex job. This column is invalid before a job is under resource pool management and control. This column is valid only when the job is under or has been under resource pool management and control. |

| Column | Type | Description |
|-----------------|------------------|--|
| lane | Text | Resource pool lane where a job is queued or executed: <ul style="list-style-type: none">• fast: fast lane.• slow: slow lane. This column is invalid before a job is under resource pool management and control. This column is valid only when the job is under or has been under resource pool management and control. |
| status | Text | Current status of a job. The value can be pending or running . |
| queue | Text | Job queuing information: <ul style="list-style-type: none">• None: The job is running.• Global: The job is queued in the global queue of the CN.• Respool: The job is queued in the resource pool.• CCN: The job is queued in the CCN. |
| used_mem | Integer | Maximum peak memory usage of a job across all DN. The unit is MB. |
| estimate_memory | Integer | Estimated memory of a job. The unit is MB. |
| used_cpu | Double precision | Average number of CPU cores occupied by a job since the job starts to run. |
| read_speed | Integer | Average logical I/O read rate of a job on all DN. The unit is KB/s. |
| write_speed | Integer | Average logical I/O write rate of a job on all DN. The unit is KB/s. |
| send_speed | Integer | Average transmit rate on all DN since a job starts to run. The unit is KB/s. |
| recv_speed | Integer | Average receive rate on all DN since a job starts to run. The unit is KB/s. |
| dn_count | Bigint | Number of DN that execute the job. |
| stream_count | Bigint | Total number of stream threads of a job on all DN. |
| pid | Bigint | ID of the backend thread |
| lwtid | Integer | Lightweight thread ID of a background thread. |
| query_id | Bigint | Query ID. |

| Column | Type | Description |
|---------------|--------|----------------------------------|
| unique_sql_id | Bigint | ID of the normalized unique SQL. |
| query | Text | Query that is being executed. |

18.3.66 GS_QUERY_RESOURCE_INFO

The **GS_QUERY_RESOURCE_INFO** view displays the resource information about all running jobs on the current DN. This parameter is supported only by clusters of version 9.1.0 or later.

NOTE

This view can be queried only on DNs. It is used only for O&M operations to locate faults. You are advised not to use this function.

Table 18-138 GS_QUERY_RESOURCE_INFO

| Column | Type | Description |
|-------------|--------|--|
| node_name | Text | Instance name, which contains only DNs |
| user_id | OID | User ID. |
| queryid | Bigint | Internal query ID used for statement execution. |
| used_mem | Int | Memory used by the statement on the current DN. The unit is MB. |
| cpu_time | Bigint | CPU time of a statement on the current DN. The unit is ms. |
| used_cpu | Double | Number of CPUs used by the statement on the current DN. |
| spill_size | Bigint | Amount of data spilled to disks on the current DN. The default value is 0 . The unit is MB. |
| read_bytes | Bigint | Number of logical read bytes used by the statement on the current DN. The unit is KB. |
| write_bytes | Bigint | Number of logical write bytes used by the statement on the current DN. The unit is KB. |
| read_count | Bigint | Number of logical reads used by the statement on the current DN. |
| write_count | Bigint | Number of logical writes used by the statement on the current DN. |
| read_speed | Int | Logical read rate used by the statement on the current DN. The unit is KB/s. |

| Column | Type | Description |
|-------------|--------|--|
| write_speed | Int | Logical write rate used by the statement on the current DN. The unit is KB/s. |
| curr_iops | Int | I/O operations per second of the statement on the current DN. It is recorded as a count in a column-store table and as a count of 10,000 in a row-store table. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DNs. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DNs. |
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |
| send_speed | Int | Network sending rate of the statement on the current DN. The unit is KB/s. |
| recv_speed | Int | Network receiving rate of the statement on the current DN. The unit is KB/s. |

18.3.67 GS_REL_IOSTAT

GS_REL_IOSTAT displays disk I/O statistics on the current node. In the current version, only one page is read or written in each read or write operation. Therefore, the number of read/write times is the same as the number of pages.

Table 18-139 GS_REL_IOSTAT columns

| Column | Type | Description |
|-----------|--------|-------------------------|
| phyrds | Bigint | Number of disk reads |
| phywrts | Bigint | Number of disk writes |
| phyblkrd | Bigint | Number of read pages |
| phyblkwrt | Bigint | Number of written pages |

18.3.68 GS_RESPOOL_RUNTIME_INFO

GS_RESPOOL_RUNTIME_INFO displays information about the running of jobs in all resource pools on the current CN.

Table 18-140 GS_RESPOOL_RUNTIME_INFO columns

| Column | Type | Description |
|-----------|------|---|
| nodegroup | Name | Name of the logical cluster the resource pool belongs to. The default cluster is installation . |
| rpname | Name | Resource pool name. |
| ref_count | Int | Number of jobs that reference the resource pool. This count includes both controlled and uncontrolled jobs. |
| fast_run | Int | Number of jobs currently running in the resource pool's fast lane. |
| fast_wait | Int | Number of jobs currently queued in the resource pool's fast lane. |
| slow_run | Int | Number of jobs currently running in the resource pool's slow lane. |
| slow_wait | Int | Number of jobs currently queued in the resource pool's slow lane. |

18.3.69 GS_RESPOOL_RESOURCE_INFO

GS_RESPOOL_RESOURCE_INFO displays job running information about all resource pools on a CN and the information about resource pool usage of an instance (CN/DN).

NOTE

On a DN, it only displays the monitoring information of the logical cluster that the DN belongs to.

Table 18-141 GS_RESPOOL_RESOURCE_INFO columns

| Column | Type | Description |
|-----------|------|--|
| nodegroup | Name | Name of the logical cluster of the resource pool. The default value is installation . |
| rpname | Name | Resource pool name |
| cgroup | Name | Name of the Cgroup associated with the resource pool |
| ref_count | Int | Number of jobs referenced by the resource pool. The number is counted regardless of whether the job is controlled by the resource pool. This parameter is valid only on CNs. |

| Column | Type | Description |
|------------|--------|--|
| fast_run | Int | Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs. |
| fast_wait | Int | Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs. |
| fast_limit | Int | Limit on the number of concurrent jobs in the fast lane in a resource pool. This parameter is valid only on CNs. |
| slow_run | Int | Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs. |
| slow_wait | Int | Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs. |
| slow_limit | Int | Limit on the number of concurrent jobs in the slow lane in a resource pool. This parameter is valid only on CNs. |
| used_cpu | Double | <p>Average number of CPUs used by the resource pool in a 5s monitoring period. The value is accurate to two decimal places.</p> <ul style="list-style-type: none">On a DN, it indicates the number of CPUs used by the resource pool on the current DN.On a CN, it indicates the total CPU usage of resource pools on all DNs. |
| cpu_limit | Int | <p>It indicates the upper limit of available CPUs for resource pools. If the CPU share is limited, this parameter indicates the available CPUs for DWS. If the CPU limit is specified, this parameter indicates the available CPUs for associated Cgroups.</p> <ul style="list-style-type: none">On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN.On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs. |

| Column | Type | Description |
|--------------|--------|--|
| used_mem | Int | Memory size used by the resource pool (unit: MB) <ul style="list-style-type: none">On a DN, it indicates the memory usage of the resource pool on the current DN.On a CN, it indicates the total memory usage of resource pools on all DNs. |
| estimate_mem | Int | Estimated memory used by the jobs running in the resource pools on the current CN. This parameter is valid only on CNs. |
| mem_limit | Int | Upper limit of available memory for the resource pool (unit: MB). <ul style="list-style-type: none">On a DN, it indicates the upper limit of available memory for the resource pool on the current DN.On a CN, it indicates the total upper limit of available memory for resource pools on all DNs. |
| read_kbytes | Bigint | Number of logical read bytes in the resource pool within a 5s monitoring period (unit: KB). <ul style="list-style-type: none">On a DN, it indicates the number of logical read bytes in the resource pool on the current DN.On a CN, it indicates the total logical read bytes of resource pools on all DNs. |
| write_kbytes | Bigint | Number of logical write bytes in the resource pool within a 5s monitoring period (unit: KB). <ul style="list-style-type: none">On a DN, it indicates the number of logical write bytes in the resource pool on the current DN.On a CN, it indicates the total logical write bytes of resource pools on all DNs. |
| read_counts | Bigint | Number of logical reads in the resource pool within a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the number of logical reads in the resource pool on the current DN.On a CN, it indicates the total number of logical reads in resource pools on all DNs. |

| Column | Type | Description |
|--------------|--------|---|
| write_counts | Bigint | Number of logical writes in the resource pool within a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the number of logical writes in the resource pool on the current DN.On a CN, it indicates the total number of logical writes in resource pools on all DNs. |
| read_speed | Double | Average rate of logical reads of the resource pool in a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the logical read rate of the resource pool on the current DN.On a CN, it indicates the overall logical read rate of resource pools on all DNs. |
| write_speed | Double | Average rate of logical writes of resource pools in a 5s monitoring period, in KB/s. <ul style="list-style-type: none">On a DN, it indicates the logical write rate of the resource pool on the current DN.On a CN, it indicates the overall logical write rate of resource pools on all DNs. |
| send_speed | Double | Average network sending rate of a resource pool in a 5-second monitoring period. The unit is KB/s. <ul style="list-style-type: none">On a DN, it indicates the network sending rate of the resource pool on the current DN.On a CN, it indicates that the cumulative sum of the network sending rates of the resource pool on all DNs. |
| recv_speed | Double | Average network receiving rate of a resource pool in a 5-second monitoring period. The unit is KB/s. <ul style="list-style-type: none">On a DN, it indicates the network receiving rate of the resource pool on the current DN.On a CN, it indicates that the cumulative sum of the network receiving rates of the resource pool on all DNs. |

18.3.70 GS_RESPOOL_MONITOR

Displays the job running information and resource usage information of all resource pools. This view can be queried only on CNs. This view is supported only by clusters of 8.2.1.100 and later versions.

Table 18-142 GS_RESPOOL_MONITOR columns

| Column | Type | Description |
|-----------------|---------|---|
| rpname | Name | Resource pool name. |
| nodegroup | Name | Name of the logical cluster the resource pool belongs to. The default value is installation . |
| cn_count | Bigint | Number of CNs in the cluster. This parameter is used to determine whether the management and control result of a single CN is proper in a multi-CN environment. |
| short_acc | Boolean | Whether to enable short query acceleration for a resource pool. |
| session_count | Bigint | Number of sessions associated with the resource pool, that is, the number of sessions initiated by users associated with the resource pool, including idle and active sessions. |
| active_count | Bigint | Number of active sessions associated with the resource pool, that is, the number of sessions that are performing queries. |
| global_wait | Bigint | Number of jobs associated with the resource pool that are queued because the number of concurrent jobs on a single CN exceeds the value of max_active_statements . |
| fast_run | Bigint | Number of jobs associated with the resource pool that are running on the fast lane. |
| fast_wait | Bigint | Number of jobs associated with the resource pool that are queued on the fast lane. |
| fast_limit | Bigint | Maximum number of concurrent jobs on the fast lane in a resource pool. |
| slow_run | Bigint | Number of jobs associated with the resource pool that are running on the slow lane. |
| slow_wait | Bigint | Number of jobs associated with the resource pool that are queued on the slow lane. |
| slow_limit | Bigint | Maximum number of concurrent jobs on the slow lane in a resource pool. |
| used_mem | Text | Average memory usage of the resource pool on all DN. The result has been formatted using pg_size_pretty . |
| estimate_memory | Text | Total estimated memory of jobs running in the resource pool. The result has been formatted using pg_size_pretty . |

| Column | Type | Description |
|-----------------|------------------|--|
| mem_limit | Text | Upper limit of the available memory in the resource pool. The result has been formatted using pg_size_pretty . |
| query_mem_limit | Name | Maximum memory that can be used by a single query in a resource pool. This parameter is used to limit the estimated query memory to prevent abnormal queuing caused by overestimation. The estimated memory is used to limit the actually used query memory. The displayed result has been formatted using pg_size_pretty . |
| used_cpu | Double precision | Average number of CPU cores occupied by a resource pool on all DNs. CPU isolation is performed by node and resource pool. If a single node contains multiple DNs, the number of CPU cores occupied by a resource pool on a single node must be multiplied by the number of DNs. |
| cpu_limit | Double precision | Average upper limit of available CPUs for a resource pool on all nodes. If CPU Time Limit is enabled, the value is the total number of available CPU cores of DWS. If CPU Usage Limit is enabled, the value is the number of available CPU cores of the associated Cgroup. |
| read_speed | Text | Average logical I/O read rate of the resource pool on all DNs. The result has been formatted using pg_size_pretty . |
| write_speed | Text | Average logical I/O write rate of the resource pool on all DNs. The result has been formatted using pg_size_pretty . |
| send_speed | Text | Average network sending rate of the resource pool on all DNs. The result has been formatted using pg_size_pretty . |
| recv_speed | Text | Average receiving rate of the resource pool on all DNs. The result has been formatted using pg_size_pretty . |

18.3.71 GS_ROW_TABLE_IO_STAT

GS_ROW_TABLE_IO_STAT displays the I/O of all row-store tables of the database on the current node. The value of each statistical column is the accumulated value since the instance was started.

Table 18-143 GS_ROW_TABLE_IO_STAT columns

| Column | Type | Description |
|------------|--------|--|
| schemaname | Name | Namespace of a table |
| relname | Name | Name of a table |
| heap_read | Bigint | Number of blocks logically read in the heap |
| heap_hit | Bigint | Number of block hits in the heap |
| idx_read | Bigint | Number of blocks logically read in the index |
| idx_hit | Bigint | Number of block hits in the index |
| toast_read | Bigint | Number of blocks logically read in the TOAST table |
| toast_hit | Bigint | Number of block hits in the TOAST table |
| tidx_read | Bigint | Number of indexes logically read in the TOAST table |
| tidx_hit | Bigint | Number of index hits in the TOAST table |

18.3.72 GS_SESSION_CPU_STATISTICS

GS_SESSION_CPU_STATISTICS displays load management information about CPU usage of ongoing complex jobs executed by the current user.

Table 18-144 GS_SESSION_CPU_STATISTICS columns

| Column | Type | Description |
|----------------|--------------------------|---|
| datid | OID | OID of the database the backend is connected to. |
| username | Name | Username logged in to the backend. |
| pid | Bigint | Backend thread ID. |
| start_time | Timestamp with time zone | Start time of statement execution. |
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DNs. The unit is ms. |

| Column | Type | Description |
|------------|------|--|
| query | Text | Statement currently being executed. |
| node_group | Text | Logical cluster of the user running the statement. |

18.3.73 GS_SESSION_MEMORY_STATISTICS

GS_SESSION_MEMORY_STATISTICS displays load management information about memory usage of ongoing complex jobs executed by the current user.

Table 18-145 GS_SESSION_MEMORY_STATISTICS columns

| Column | Type | Description |
|-----------------|--------------------------|--|
| datid | OID | OID of the database the backend is connected to. |
| username | Name | Username logged in to the backend. |
| pid | Bigint | Backend thread ID. |
| start_time | Timestamp with time zone | Start time of statement execution. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DNs, in MB |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DNs, in MB |
| spill_info | Text | Spill information for the statement on all DNs. The options are:
None : The statement has not been spilled to disks on any DNs.
All : The statement has been spilled to disks on all DNs.
[a:b] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DNs. |
| query | Text | Statement currently being executed. |
| node_group | Text | Logical cluster of the user running the statement. |

18.3.74 GS_SQL_COUNT

GS_SQL_COUNT displays statistics about the five types of statements (**SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**) executed on the current node of

the database, including the number of execution times, response time (the maximum, minimum, average, and total response time of the other four types of statements except the **MERGE INTO** statement, in microseconds), and the number of execution times of **DDL**, **DML**, and **DCL** statements.

The classification of **DDL**, **DML**, and **DCL** statements in the **GS_SQL_COUNT** view is slightly different from that of the SQL syntax. The details are as follows:

- User-related statements, such as **CREATE/ALTER/DROP USER** and **CREATE/ALTER/DROP ROLE**, are of the DCL type.
- Transaction-related statements such as **BEGIN/COMMIT/SET CONSTRAINTS/ROLLBACK/SAVEPOINT/START** are of the DCL type.
- **ALTER SYSTEM KILL SESSION** is equivalent to the **SELECT pg_terminate_backend()** statement and is of the DML type.

The classification of other statements is similar to the definition in the SQL syntax.

When a common user queries the **GS_SQL_COUNT** view, only the statistics of this user in the current node can be viewed. When a user with the administrator permissions queries the **GS_SQL_COUNT** view, the statistics of all users in the current node can be viewed. When the cluster or the node is restarted, the statistics are cleared and the counting restarts. The counting is based on the number of queries received by the node, including the queries performed inside the cluster. Statistics about the **GS_SQL_COUNT** view are collected only on CNs, and SQL statements sent from other CNs are not collected. No result is returned when you query the view on a DN.

Table 18-146 GS_SQL_COUNT columns

| Column | Type | Description |
|---------------------|--------|---|
| node_name | Name | Node name |
| user_name | Name | Username |
| select_count | Bigint | Number of SELECT statements. |
| update_count | Bigint | Number of UPDATE statements |
| insert_count | Bigint | Number of INSERT statements |
| delete_count | Bigint | Number of DELETE statements |
| mergeinto_count | Bigint | Number of MERGE INTO statements |
| ddl_count | Bigint | Number of DDL statements |
| dml_count | Bigint | Number of DML statements |
| dcl_count | Bigint | Number of DCL statements |
| total_select_elapse | Bigint | Total response time of SELECT statements |
| avg_select_elapse | Bigint | Average response time of SELECT statements |

| Column | Type | Description |
|---------------------|--------|---|
| max_select_elapse | Bigint | Maximum response time of SELECT statements |
| min_select_elapse | Bigint | Minimum response time of SELECT statements |
| total_update_elapse | Bigint | Total response time of UPDATE statements |
| avg_update_elapse | Bigint | Average response time of UPDATE statements |
| max_update_elapse | Bigint | Maximum response time of UPDATE statements |
| min_update_elapse | Bigint | Minimum response time of UPDATE statements |
| total_delete_elapse | Bigint | Total response time of DELETE statements |
| avg_delete_elapse | Bigint | Average response time of DELETE statements |
| max_delete_elapse | Bigint | Maximum response time of DELETE statements |
| min_delete_elapse | Bigint | Minimum response time of DELETE statements |
| total_insert_elapse | Bigint | Total response time of INSERT statements |
| avg_insert_elapse | Bigint | Average response time of INSERT statements |
| max_insert_elapse | Bigint | Maximum response time of INSERT statements |
| min_insert_elapse | Bigint | Minimum response time of INSERT statements |

18.3.75 GS_STAT_DB_CU

GS_STAT_DB_CU displays CU hits of each database in each node of a cluster. You can clear it using **gs_stat_reset()**.

Table 18-147 GS_STAT_DB_CU columns

| Column | Type | Description |
|------------|------|---------------|
| node_name1 | Text | Node name |
| db_name | Text | Database name |

| Column | Type | Description |
|---------------|--------|-----------------------------------|
| mem_hit | Bigint | Number of memory hits |
| hdd_sync_read | Bigint | Number of disk synchronous reads |
| hdd_asyn_read | Bigint | Number of disk asynchronous reads |

18.3.76 GS_STAT_SESSION_CU

GS_STAT_SESSION_CU displays the CU hit rate of running sessions on each node in a cluster. This data about a session is cleared when you exit this session or restart the cluster.

Table 18-148 GS_STAT_SESSION_CU columns

| Column | Type | Description |
|---------------|---------|-----------------------------------|
| node_name1 | Text | Node name |
| mem_hit | Integer | Number of memory hits |
| hdd_sync_read | Integer | Number of disk synchronous reads |
| hdd_asyn_read | Integer | Number of disk asynchronous reads |

18.3.77 GS_TABLE_CHANGE_STAT

GS_TABLE_CHANGE_STAT displays the changes of all tables (excluding foreign tables) of the database on the current node. The value of each column that indicates the number of times is the accumulated value since the instance was started.

Table 18-149 GS_TABLE_CHANGE_STAT columns

| Column | Type | Description |
|--------------|--------------------------|--|
| schemaname | Name | Namespace of a table |
| relname | Name | Table name |
| last_vacuum | Timestamp with time zone | Time when the last VACUUM operation is performed manually |
| vacuum_count | Bigint | Number of times of manually performing the VACUUM operation |

| Column | Type | Description |
|-------------------|--------------------------|---|
| last_autovacuum | Timestamp with time zone | Time when the last VACUUM operation is performed automatically |
| autovacuum_count | Bigint | Number of times of automatically performing the VACUUM operation |
| last_analyze | Timestamp with time zone | Time when the last non-automatic rolling analysis is performed. |
| analyze_count | Bigint | Number of times of performing the ANALYZE operation (both manually and automatically) |
| last_autoanalyze | Timestamp with time zone | Time when the last automatic rolling analysis is performed. |
| autoanalyze_count | Bigint | Number of times of automatically performing the ANALYZE operation |
| last_change | Bigint | Time when the last modification (INSERT , UPDATE , or DELETE) is performed |

18.3.78 GS_TABLE_STAT

GS_TABLE_STAT displays statistics about all tables (excluding foreign tables) of the database on the current node. The values of **live_tuples** and **dead_tuples** are real-time values, and the values of other statistical columns are accumulated values since the instance was started.

Table 18-150 GS_TABLE_STAT columns

| Column | Type | Description |
|------------------|--------|---|
| schemaname | Name | Namespace of a table |
| relname | Name | Table name |
| seq_scan | Bigint | Number of sequential scans. For a partitioned table, the sum of the number of scans of each partition is displayed. |
| seq_tuple_read | Bigint | Number of rows scanned in sequence. |
| index_scan | Bigint | Number of index scans. |
| index_tuple_read | Bigint | Number of rows scanned by the index. |
| tuple_inserted | Bigint | Number of rows inserted. |

| Column | Type | Description |
|-------------------|--------|---|
| tuple_updated | Bigint | Number of rows updated. |
| tuple_deleted | Bigint | Number of rows deleted. |
| tuple_hot_updated | Bigint | Number of rows with HOT updates. |
| live_tuples | Bigint | Number of live tuples. Query the view on the CN. If ANALYZE is executed, the total number of live tuples in the table is displayed. Otherwise, 0 is displayed. This indicator applies only to row-store tables. |
| dead_tuples | Bigint | Number of dead tuples. Query the view on the CN. If ANALYZE is executed, the total number of dead tuples in the table is displayed. Otherwise, 0 is displayed. This indicator applies only to row-store tables. |

18.3.79 GS_TOTAL_NODEGROUP_MEMORY_DETAIL

GS_TOTAL_NODEGROUP_MEMORY_DETAIL displays statistics about memory usage of the logical cluster that the current database belongs to in the unit of MB.

Table 18-151 GS_TOTAL_NODEGROUP_MEMORY_DETAIL columns

| Column | Type | Description |
|--------|------|----------------------|
| ngname | Text | Logical cluster name |

| Column | Type | Description |
|-------------|---------|--|
| memorytype | Text | Memory type. The value can be: <ul style="list-style-type: none">• ng_total_memory: total memory of the logical cluster• ng_used_memory: memory usage of the logical cluster• ng_estimate_memory: estimated memory usage of the logical cluster• ng_foreignrp_memsize: total memory of the external resource pool of the logical cluster• ng_foreignrp_usedsize: memory usage of the external resource pool of the logical instance• ng_foreignrp_peaksize: peak memory usage of the external resource pool of the logical cluster• ng_foreignrp_mempct: percentage of the external resource pool of the logical cluster to the total memory of the logical cluster• ng_foreignrp_estmsize: estimated memory usage of the external resource pool of the logical cluster |
| memorybytes | Integer | Size of allocated memory-typed memory |

18.3.80 GS_USER_MONITOR

GS_USER_MONITOR displays all users' job running and resource usage information. This view can be queried only on CNs. This view is supported only by clusters of 8.2.1.100 and later versions.

Table 18-152 GS_USER_MONITOR columns

| Column | Type | Description |
|---------------|--------|---|
| username | Name | Username |
| rpname | Name | Name of the resource pool associated with the user |
| nodegroup | Name | Name of the logical cluster the resource pool belongs to. The default value is installation . |
| session_count | Bigint | Number of sessions initiated by the user, including idle and active sessions |
| active_count | Bigint | Number of active sessions initiated by the user, that is, the number of sessions that are performing queries. |
| global_wait | Bigint | Number of jobs that are queued because the number of concurrent jobs on a single CN exceeds the value of max_active_statements . |

| Column | Type | Description |
|-------------------|------------------|--|
| fast_run | Bigint | Number of jobs that are running on the fast lane of the resource pool among all jobs executed by the user. |
| fast_wait | Bigint | Number of jobs queued in the fast lane of the resource pool among all jobs executed by the user. |
| slow_run | Bigint | Number of jobs that are running on the slow lane of the resource pool among all jobs executed by the user. |
| slow_wait | Bigint | Number of jobs queued in the slow lane of the resource pool among all jobs executed by the user. |
| used_mem | Bigint | Average memory used by a user on all DNs, in MB. |
| estimate_memory | Bigint | Total estimated memory used by running jobs, in MB. |
| used_cpu | Double precision | Average number of CPU cores used by a user on all DNs. If a single node contains multiple DNs, the number of CPU cores used by a user on the node must be multiplied by the number of DNs. |
| read_speed | Bigint | Average logical I/O read rate of a user on all DNs, in KB/s. |
| write_speed | Bigint | Average logical I/O write rate of a user on all DNs, in KB/s. |
| send_speed | Bigint | Average data sending rate of a user on all DNs, in KB/s. |
| recv_speed | Bigint | Average data receiving rate of a user on all DNs, in KB/s. |
| used_space | Bigint | Used space of user permanent tables, in KB. |
| space_limit | Bigint | Maximum space that can be used by user permanent tables, in KB. The value -1 indicates that the space size is not limited. |
| used_temp_space | Bigint | Used space of user temporary tables, in KB. |
| temp_space_limit | Bigint | Maximum space that can be used by user temporary tables, in KB. The value -1 indicates that the space size is not limited. |
| used_spill_space | Bigint | Used space for flushing intermediate result sets, in KB. |
| spill_space_limit | Bigint | Maximum space that can be used for flushing intermediate result sets, in KB. The value -1 indicates that the space size is not limited. |

18.3.81 GS_USER_TRANSACTION

GS_USER_TRANSACTION provides transaction information about users on a single CN. The database records the number of times that each user commits and rolls back transactions and the response time of transaction commitment and rollback, in microseconds.

Table 18-153 GS_USER_TRANSACTION columns

| Column | Type | Description |
|------------------|--------|-----------------------|
| username | Name | Username |
| commit_counter | Bigint | Number of the commits |
| rollback_counter | Bigint | Number of rollbacks |
| resp_min | Bigint | Minimum response time |
| resp_max | Bigint | Maximum response time |
| resp_avg | Bigint | Average response time |
| resp_total | Bigint | Total response time |

18.3.82 GS_VIEW_DEPENDENCY

GS_VIEW_DEPENDENCY allows you to query the direct dependencies of all views visible to the current user.

Table 18-154 GS_VIEW_DEPENDENCY columns

| Column | Type | Description |
|--------------|------|---|
| objschema | Name | View space name |
| objname | Name | View name |
| refobjschema | Name | Name of the space where the dependent object resides |
| refobjname | Name | Name of a dependent object |
| relobjkind | Char | Type of a dependent object <ul style="list-style-type: none">• r: table• v: view |

18.3.83 GS_VIEW_DEPENDENCY_PATH

GS_VIEW_DEPENDENCY_PATH allows you to query the direct dependencies of all views visible to the current user. If the base table on which the view depends exists and the dependency between views at different levels is normal, you can use this view to query the dependency between views at different levels starting from the base table.

Table 18-155 GS_VIEW_DEPENDENCY_PATH columns

| Column | Type | Description |
|--------------|------|--|
| objschema | Name | View space name |
| objname | Name | View name |
| refobjschema | Name | Name of the space where the dependent object resides |
| refobjname | Name | Name of a dependent object |
| path | Text | Dependency path |

18.3.84 GS_VIEW_INVALID

GS_VIEW_INVALID queries all unavailable views visible to the current user.

If the basic table, function, or synonym on which the view depends is abnormal, the **validtype** column of the view is displayed as **invalid**. If the system object on which the view depends changes during an upgrade, the **validtype** column of the view is displayed as **invalidInUpgrade**.

Table 18-156 GS_VIEW_INVALID columns

| Column | Type | Description |
|------------|------|------------------------|
| OID | OID | OID of the view |
| schemaname | Name | View space name |
| viewname | Name | Name of the view |
| viewowner | Name | Owner of the view |
| definition | Text | Definition of the view |
| validtype | Text | View validity flag |

| Column | Type | Description |
|-------------------|--------------------------|---|
| last_invalid_time | Timestamp with time zone | Time when a view is invalid. This column is available only in clusters of version 9.1.0.200 or later. |

18.3.85 GS_WAIT_EVENTS

GS_WAIT_EVENTS displays statistics about waiting status and events on the current node.

The values of statistical columns in this view are accumulated only when the **enable_track_wait_event** GUC parameter is set to **on**. If **enable_track_wait_event** is set to **off** during statistics measurement, the statistics will no longer be accumulated, but the existing values are not affected. If **enable_track_wait_event** is **off**, 0 row is returned when this view is queried.

Table 18-157 GS_WAIT_EVENTS columns

| Column | Type | Description |
|-----------------|--------|--|
| nodename | Name | Node name |
| type | Text | Event type, which can be STATUS , LOCK_EVENT , LWLOCK_EVENT , or IO_EVENT |
| event | Text | Event name. For details, see PG_THREAD_WAIT_STATUS . |
| wait | Bigint | Number of times an event occurs. This column and all the columns below are values accumulated during process running. |
| failed_wait | Bigint | Number of waiting failures. In the current version, this column is used only for counting timeout errors and waiting failures of locks such as LOCK and LWLOCK . |
| total_wait_time | Bigint | Total duration of the event |
| avg_wait_time | Bigint | Average duration of the event |
| max_wait_time | Bigint | Maximum wait time of the event |
| min_wait_time | Bigint | Minimum wait time of the event |

In the current version, for events whose **type** is **LOCK_EVENT**, **LWLOCK_EVENT**, or **IO_EVENT**, the display scope of **GS_WAIT_EVENTS** is the same as that of the corresponding events in the **PG_THREAD_WAIT_STATUS** view.

For events whose **type** is **STATUS**, **GS_WAIT_EVENTS** displays the following waiting status columns. For details, see the **PG_THREAD_WAIT_STATUS** view.

- acquire lwlock
- acquire lock
- wait io
- wait pooler get conn
- wait pooler abort conn
- wait pooler clean conn
- wait transaction sync
- wait wal sync
- wait data sync
- wait producer ready
- create index
- analyze
- vacuum
- vacuum full
- gtm connect
- gtm begin trans
- gtm commit trans
- gtm rollback trans
- gtm create sequence
- gtm alter sequence
- gtm get sequence val
- gtm set sequence val
- gtm drop sequence
- gtm rename sequence

18.3.86 GS_WLM_OPERATOR_INFO

This view displays the execution information about operators in the query statements that have been executed on the current CN. The information comes from the system catalog **dbms_om**. [gs_wlm_operator_info](#).

NOTICE

- The schema of the **GS_WLM_OPERATOR_INFO** view is **pg_catalog**.
 - The **GS_WLM_OPERATOR_INFO** view can be queried only in the **postgres** database. If the view is queried in other databases, an error is reported.
-

Table 18-158 GS_WLM_OPERATOR_INFO columns

| Column | Type | Description |
|---------------------|--------------------------|--|
| nodename | Text | Name of the CN where the statement is executed |
| queryid | Bigint | Internal query_id used for statement execution |
| pid | Bigint | Backend thread ID |
| plan_node_id | Integer | plan_node_id of the execution plan of a query |
| plan_node_name | Text | Name of the operator corresponding to plan_node_id |
| start_time | Timestamp with time zone | Time when an operator starts to process the first data record |
| duration | Bigint | Total execution time of an operator. The unit is ms. |
| query_dop | Integer | Degree of parallelism (DOP) of the current operator |
| estimated_rows | Bigint | Number of rows estimated by the optimizer |
| tuple_processed | Bigint | Number of elements returned by the current operator |
| min_peak_memory | Integer | Minimum peak memory used by the current operator on all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum peak memory used by the current operator on all DN. The unit is MB. |
| average_peak_memory | Integer | Average peak memory used by the current operator on all DN. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of the current operator among DN |
| min_spill_size | Integer | Minimum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |

| Column | Type | Description |
|------------------|---------|--|
| min_cpu_time | Bigint | Minimum execution time of the operator on all DN. The unit is ms. |
| max_cpu_time | Bigint | Maximum execution time of the operator on all DN. The unit is ms. |
| total_cpu_time | Bigint | Total execution time of the operator on all DN. The unit is ms. |
| cpu_skew_percent | Integer | Skew of the execution time among DN. |
| warning | Text | Warning. The following warnings are displayed:
1. Sort/SetOp/HashAgg/HashJoin spill
2. The max spill size exceeds the alarm size 256MB
3. The max broadcast size exceeds the alarm size 100MB
4. Early spill
5. Spill times is greater than 3
6. Spill on memory adaptive
7. Hash table conflict
8. NestLoop in hashjoin |

18.3.87 GS_WLM_OPERATOR_HISTORY

GS_WLM_OPERATOR_HISTORY displays the records of operators in jobs that have been executed by the current user on the current CN.

This view is used to query data from DWS. Data in the database is cleared periodically. If the GUC parameter **enable_resource_record** is set to **on**, records in the view will be dumped to the system catalog **GS_WLM_OPERATOR_INFO** every 3 minutes and deleted from the view. If **enable_resource_record** is set to **off**, the records will be deleted from the view after the retention period expires. The recorded data is the same as that described in [Table 18-159](#).

Table 18-159 GS_WLM_OPERATOR_INFO columns

| Column | Type | Description |
|----------|--------|--|
| nodename | Text | Name of the CN where the statement is executed |
| queryid | Bigint | Internal query_id used for statement execution |
| pid | Bigint | Backend thread ID |

| Column | Type | Description |
|---------------------|--------------------------|--|
| plan_node_id | Integer | plan_node_id of the execution plan of a query |
| plan_node_name | Text | Name of the operator corresponding to plan_node_id |
| start_time | Timestamp with time zone | Time when an operator starts to process the first data record |
| duration | Bigint | Total execution time of an operator. The unit is ms. |
| query_dop | Integer | Degree of parallelism (DOP) of the current operator |
| estimated_rows | Bigint | Number of rows estimated by the optimizer |
| tuple_processed | Bigint | Number of elements returned by the current operator |
| min_peak_memory | Integer | Minimum peak memory used by the current operator on all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum peak memory used by the current operator on all DN. The unit is MB. |
| average_peak_memory | Integer | Average peak memory used by the current operator on all DN. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of the current operator among DN |
| min_spill_size | Integer | Minimum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_cpu_time | Bigint | Minimum execution time of the operator on all DN. The unit is ms. |
| max_cpu_time | Bigint | Maximum execution time of the operator on all DN. The unit is ms. |
| total_cpu_time | Bigint | Total execution time of the operator on all DN. The unit is ms. |

| Column | Type | Description |
|------------------|---------|--|
| cpu_skew_percent | Integer | Skew of the execution time among DN. |
| warning | Text | Warning. The following warnings are displayed:
1. Sort/SetOp/HashAgg/HashJoin spill
2. The max spill size exceeds the alarm size 256MB
3. The max broadcast size exceeds the alarm size 100MB
4. Early spill
5. Spill times is greater than 3
6. Spill on memory adaptive
7. Hash table conflict
8. NestLoop in hashjoin |

18.3.88 GS_WLM_OPERATOR_STATISTICS

GS_WLM_OPERATOR_STATISTICS displays the operators of the jobs that are being executed by the current user.

Table 18-160 GS_WLM_OPERATOR_STATISTICS columns

| Column | Type | Description |
|----------------|--------------------------|---|
| queryid | Bigint | Internal query_id used for statement execution |
| pid | Bigint | ID of the backend thread |
| plan_node_id | Integer | plan_node_id of the execution plan of a query |
| plan_node_name | Text | Name of the operator corresponding to plan_node_id . The maximum length of the operator name is 127 characters (excluding format characters such as spaces). |
| start_time | Timestamp with time zone | Time when the operator starts to be executed for the first time. |
| duration | Bigint | Total execution time of the operator from the start to the end, in milliseconds. |
| status | Text | Execution status of the current operator. The value can be waiting , running , or finished . |
| query_dop | Integer | DOP of the current operator |

| Column | Type | Description |
|---------------------|---------|--|
| estimated_rows | Bigint | Number of rows estimated by the optimizer. If the number of returned estimated rows exceeds int64_max, int64_max is displayed. |
| tuple_processed | Bigint | Total number of elements returned by the current operator on all DN. If the estimated number of returned rows exceeds int64_max, int64_max is displayed. |
| min_peak_memory | Integer | Minimum peak memory used by the current operator on all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum peak memory used by the current operator on all DN. The unit is MB. |
| average_peak_memory | Integer | Average peak memory used by the current operator on all DN. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of the current operator among DN |
| min_spill_size | Integer | Minimum logical spilled data among all DN when a spill occurs, in MB. The default value is 0. |
| max_spill_size | Integer | Maximum logical spilled data among all DN when a spill occurs, in MB. The default value is 0. |
| average_spill_size | Integer | Average logical spilled data among all DN when a spill occurs, in MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_cpu_time | Bigint | Minimum execution time of the operator on all DN. The unit is ms. |
| max_cpu_time | Bigint | Maximum execution time of the operator on all DN. The unit is ms. |
| total_cpu_time | Bigint | Total execution time of the operator on all DN. The unit is ms. |
| cpu_skew_percent | Integer | Skew of the execution time among DN. |

| Column | Type | Description |
|-------------------|---------|--|
| warning | Text | Warning. The following warnings are displayed:
1. Sort/SetOp/HashAgg/HashJoin spill
2. The max spill size exceeds the alarm size 256MB
3. The max broadcast size exceeds the alarm size 100MB
4. Early spill
5. Spill times is greater than 3
6. Spill on memory adaptive
7. Hash table conflict
8. NestLoop in hashjoin |
| parent_id | Integer | Parent node ID of the operator node. |
| exec_count | Integer | Maximum number of times that the operator node can be executed on all DNs. |
| progress | Text | Progress information of the operator. For the first operator, it is the overall progress of the job. For other operators, it is the progress of the current operator. |
| min_net_size | Bigint | Minimum network communication data volume (KB) of the operator on all DNs. It mainly applies to network operators. |
| max_net_size | Bigint | Maximum network communication data volume (KB) of the operator on all DNs. It mainly applies to network operators. |
| total_net_size | Bigint | Total network communication data volume (KB) of the operator on all DNs. It mainly applies to network operators. |
| min_read_bytes | Bigint | Minimum amount of data read by the operator from disks on all DNs. The unit is KB. |
| max_read_bytes | Bigint | Maximum amount of data read by the operator from disks on all DNs. The unit is KB. |
| total_read_bytes | Bigint | Total amount of data read by the operator from disks on all DNs, in KB. |
| min_write_bytes | Bigint | Minimum amount of data written by the operator to disks on all DNs. The unit is KB. |
| max_write_bytes | Bigint | Maximum amount of data written by the operator to disks on all DNs. The unit is KB. |
| total_write_bytes | Bigint | Total amount of data written by the operator to disks on all DNs, in KB. |

18.3.89 GS_WLM_SESSION_INFO

This view displays the execution information about the query statements that have been executed on the current CN. The information comes from the system catalog **dbms_om.gs_wlm_session_info**.

NOTICE

- The schema of the **GS_WLM_SESSION_INFO** view is **pg_catalog**.
- The **GS_WLM_SESSION_INFO** view can be queried only in the **postgres** database. If the view is queried in other databases, an error is reported.

Table 18-162 lists the columns in the **GS_WLM_SESSION_INFO** view.

Table 18-161 GS_WLM_SESSION_HISTORY columns

| Column | Type | Description |
|------------------|---------|--|
| datid | OID | OID of the database this backend is connected to |
| dbname | Text | Name of the database the backend is connected to |
| schemaname | Text | Schema name |
| nodename | Text | Name of the CN where the statement is run |
| username | Text | User name used for connecting to the backend |
| application_name | Text | Name of the application that is connected to the backend |
| client_addr | inet | IP address of the client connected to this backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used |

| Column | Type | Description |
|---------------------|--------------------------|--|
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |
| block_time | Bigint | Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is ms. |
| start_time | Timestamp with time zone | Time when the statement starts to be run |
| finish_time | Timestamp with time zone | Time when the statement execution ends |
| duration | Bigint | Execution time of a statement. The unit is ms. |
| estimate_total_time | Bigint | Estimated execution time of a statement. The unit is ms. |
| status | Text | Final statement execution status. Its value can be finished (normal) or aborted (abnormal). The statement status here is the execution status of the database server. If the statement is successfully executed on the database server but an error is reported in the result set, the statement status is finished . |
| abort_info | Text | Exception information displayed if the final statement execution status is aborted . |
| resource_pool | Text | Resource pool used by the user |
| control_group | Text | Cgroup used by the statement |
| estimate_memory | Integer | Estimated memory used by a statement on a single instance. The unit is MB. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DN. The unit is MB. |
| average_peak_memory | Integer | Average memory usage during statement execution. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of a statement among DN. |

| Column | Type | Description |
|----------------------|---------|--|
| spill_info | Text | Statement spill information on all DNs.
None indicates that the statement has not been spilled to disks on any DNs.
All : The statement has been spilled to disks on all DNs.
[<i>a</i>:<i>b</i>] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DNs. |
| min_spill_size | Integer | Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 . |
| max_spill_size | Integer | Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 . |
| average_spill_size | Integer | Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 . |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_dn_time | Bigint | Minimum execution time of a statement across all DNs. The unit is ms. |
| max_dn_time | Bigint | Maximum execution time of a statement across all DNs. The unit is ms. |
| average_dn_time | Bigint | Average execution time of a statement across all DNs. The unit is ms. |
| dn_time_skew_percent | Integer | Execution time skew of a statement among DNs. |
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DNs. The unit is ms. |
| cpu_skew_percent | Integer | CPU time skew of a statement among DNs. |
| min_peak_iops | Integer | Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |

| Column | Type | Description |
|-------------------|---------|---|
| max_peak_iops | Integer | Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| average_peak_iops | Integer | Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| iops_skew_percent | Integer | I/O skew across DNs. |
| warning | Text | Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed:
<ol style="list-style-type: none">1. The max spill size exceeds the alarm size 256MB2. The max broadcast size exceeds the alarm size 100MB3. Early spill4. Spill times is greater than 35. Spill on memory adaptive6. Hash table conflict7. NestLoop in hashjoin |
| queryid | Bigint | Internal query ID used for statement execution |
| query | Text | Statement to be executed. A maximum of 64 KB of strings can be retained. |
| query_plan | Text | Execution plan of a statement.
Specification restrictions:
<ol style="list-style-type: none">1. Execution plans are displayed only for DML statements.2. In 8.2.1.100 and later versions, the number of data binding times is added to the execution plans of Parse Bind Execute (PBE) statements to facilitate statement analysis. The number of data binding times is displayed in the format of PBE bind times: Times. |
| node_group | Text | Logical cluster of the user running the statement |
| pid | Bigint | PID of the backend thread of the statement |
| lane | Text | Fast/Slow lane where the statement is executed |

| Column | Type | Description |
|---------------------|--------|---|
| unique_sql_id | Bigint | ID of the normalized unique SQL. |
| session_id | Text | Unique identifier of a session in the database system. Its format is session_start_time.tid.node_name . |
| min_read_bytes | Bigint | Minimum I/O read bytes of a statement across all DN. The unit is byte. |
| max_read_bytes | Bigint | Maximum I/O read bytes of a statement across all DN. The unit is byte. |
| average_read_bytes | Bigint | Average I/O read bytes of a statement across all DN. |
| min_write_bytes | Bigint | Minimum I/O write bytes of a statement across all DN. |
| max_write_bytes | Bigint | Maximum I/O write bytes of a statement across all DN. |
| average_write_bytes | Bigint | Average I/O write bytes of a statement across all DN. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DN. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DN. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| stmt_type | Text | Query type corresponding to the statement. |
| except_info | Text | Information about the exception rule triggered by the statement. |
| unique_plan_id | Bigint | ID of the normalized unique plan. |
| sql_hash | Text | Normalized SQL hash. |
| plan_hash | Text | Normalized plan hash. |
| use_plan_baseline | Text | Indicates whether the bound plan is used for executing the current statement. If it is used, the name of the plan_baseline column in pg_plan_baseline is displayed. |
| outline_name | Text | Name of the outline used for the statement plan. |

| Column | Type | Description |
|-----------------------------|--------------|---|
| loader_status | Text | The JSON string for storing import and export service information is as follows.
<ol style="list-style-type: none">1. address: indicates the IP address of the peer cluster. The port number is displayed for the source cluster.2. direction: indicates the import and export service type. The value can be gds to file, gds from file, gds to pipe, gds from pipe, copy from or copy to.3. min/max/total_lines/bytes: indicates the minimum value, maximum value, total lines, and bytes of the import and export statements on all DNs. |
| parse_time | Bigint | Total parsing time before the statement is queued (including lexical and syntax parsing, optimization rewriting, and plan generation time), in milliseconds. This column is available only in clusters of version 8.3.0.100 or later. |
| disk_cache_hit_ratio | numeric(5,2) | Disk cache hit rate. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_read_size | Bigint | Total size of data read from disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_write_size | Bigint | Total size of data written to disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_size | Bigint | Total size of data read remotely from OBS due to disk cache read failure, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_time | Bigint | Total number of times data is read remotely from OBS due to disk cache read failure. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| vfs_scan_bytes | Bigint | Total number of bytes scanned by the OBS virtual file system in response to upper-layer requests, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|--|
| vfs_remote_read_bytes | Bigint | Total number of bytes actually read from OBS by the OBS virtual file system, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| preload_submit_time | Bigint | Total time for submitting I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_time | Bigint | Total time for waiting for I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_count | Bigint | Total number of times that the prefetching process waits for I/O requests. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| disk_cache_loaded_time | Bigint | Total time for reading from disk cache, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_conflict_count | Bigint | Number of times a block in the disk cache produces a hash conflict. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_error_count | Bigint | Number of disk cache read failures. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|---|
| disk_cache_error_code | Bigint | <p>Error code for disk cache read failures. Multiple error codes may be generated. If the disk cache fails to be read, OBS remote read is initiated and cache blocks are rewritten. The error code types are as follows: This column only applies to OBS 3.0 tables and foreign tables.</p> <ul style="list-style-type: none">• 1: A hash conflict occurs in the disk cache block.• 2: The generation time of the disk cache block is later than that of the OldestXmin transaction.• 4: Invoking the pread system when reading cache files from the disk cache failed.• 8: The data version of the disk cache block does not match.• 16: The version of the data written to the write cache does not match the latest version.• 32: Opening the cache file corresponding to the cache block failed.• 64: The size of the data read from the disk cache does not match.• 128: The CSN recorded in the disk cache block does not match. |
| obs_io_req_avg_rtt | Bigint | Average Round Trip Time (RTT) for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_avg_latency | Bigint | Average delay for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_1s | Bigint | Number of OBS I/O requests with a latency exceeding 1 second. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_10s | Bigint | Number of OBS I/O requests with a latency exceeding 10 seconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_count | Bigint | Total number of OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|-----------------------------|--------|---|
| obs_io_req_retry_count | Bigint | Total number of retries for OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_rate_limit_count | Bigint | Total number of times OBS I/O requests are flow-controlled. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

18.3.90 GS_WLM_SESSION_HISTORY

GS_WLM_SESSION_HISTORY displays load management information about a completed job executed by the current user on the current CN. The view is used to query data from DWS. The view returns the data queried from the **GS_WLM_SESSION_INFO** table within 3 minutes only if the GUC parameter **enable_resource_track** is set to **on**.

NOTICE

The **GS_WLM_SESSION_HISTORY** view can be queried only in the **postgres** database. If the view is queried in other databases, an error is reported.

Table 18-162 GS_WLM_SESSION_HISTORY columns

| Column | Type | Description |
|------------------|------|--|
| datid | OID | OID of the database this backend is connected to |
| dbname | Text | Name of the database the backend is connected to |
| schemaname | Text | Schema name |
| nodename | Text | Name of the CN where the statement is run |
| username | Text | User name used for connecting to the backend |
| application_name | Text | Name of the application that is connected to the backend |
| client_addr | inet | IP address of the client connected to this backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |

| Column | Type | Description |
|---------------------|--------------------------|--|
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used |
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |
| block_time | Bigint | Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is ms. |
| start_time | Timestamp with time zone | Time when the statement starts to be run |
| finish_time | Timestamp with time zone | Time when the statement execution ends |
| duration | Bigint | Execution time of a statement. The unit is ms. |
| estimate_total_time | Bigint | Estimated execution time of a statement. The unit is ms. |
| status | Text | Final statement execution status. Its value can be finished (normal) or aborted (abnormal). The statement status here is the execution status of the database server. If the statement is successfully executed on the database server but an error is reported in the result set, the statement status is finished . |
| abort_info | Text | Exception information displayed if the final statement execution status is aborted . |
| resource_pool | Text | Resource pool used by the user |
| control_group | Text | Cgroup used by the statement |
| estimate_memory | Integer | Estimated memory used by a statement on a single instance. The unit is MB. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DNs. The unit is MB. |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DNs. The unit is MB. |

| Column | Type | Description |
|----------------------|---------|--|
| average_peak_memory | Integer | Average memory usage during statement execution. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of a statement among DN. |
| spill_info | Text | Statement spill information on all DN.
None indicates that the statement has not been spilled to disks on any DN.
All : The statement has been spilled to disks on all DN.
[a:b] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DN. |
| min_spill_size | Integer | Minimum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DN when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_dn_time | Bigint | Minimum execution time of a statement across all DN. The unit is ms. |
| max_dn_time | Bigint | Maximum execution time of a statement across all DN. The unit is ms. |
| average_dn_time | Bigint | Average execution time of a statement across all DN. The unit is ms. |
| dn_time_skew_percent | Integer | Execution time skew of a statement among DN. |
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DN. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DN. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DN. The unit is ms. |
| cpu_skew_percent | Integer | CPU time skew of a statement among DN. |

| Column | Type | Description |
|-------------------|---------|---|
| min_peak_iops | Integer | Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| max_peak_iops | Integer | Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| average_peak_iops | Integer | Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| iops_skew_percent | Integer | I/O skew across DNs. |
| warning | Text | Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed:
<ol style="list-style-type: none">1. The max spill size exceeds the alarm size 256MB2. The max broadcast size exceeds the alarm size 100MB3. Early spill4. Spill times is greater than 35. Spill on memory adaptive6. Hash table conflict7. NestLoop in hashjoin |
| queryid | Bigint | Internal query ID used for statement execution |
| query | Text | Statement to be executed. A maximum of 64 KB of strings can be retained. |
| query_plan | Text | Execution plan of a statement.
Specification restrictions:
<ol style="list-style-type: none">1. Execution plans are displayed only for DML statements.2. In 8.2.1.100 and later versions, the number of data binding times is added to the execution plans of Parse Bind Execute (PBE) statements to facilitate statement analysis. The number of data binding times is displayed in the format of PBE bind times: Times. |
| node_group | Text | Logical cluster of the user running the statement |

| Column | Type | Description |
|---------------------|--------|---|
| pid | Bigint | PID of the backend thread of the statement |
| lane | Text | Fast/Slow lane where the statement is executed |
| unique_sql_id | Bigint | ID of the normalized unique SQL. |
| session_id | Text | Unique identifier of a session in the database system. Its format is session_start_time.tid.node_name . |
| min_read_bytes | Bigint | Minimum I/O read bytes of a statement across all DNs. The unit is byte. |
| max_read_bytes | Bigint | Maximum I/O read bytes of a statement across all DNs. The unit is byte. |
| average_read_bytes | Bigint | Average I/O read bytes of a statement across all DNs. |
| min_write_bytes | Bigint | Minimum I/O write bytes of a statement across all DNs. |
| max_write_bytes | Bigint | Maximum I/O write bytes of a statement across all DNs. |
| average_write_bytes | Bigint | Average I/O write bytes of a statement across all DNs. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DNs. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DNs. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| stmt_type | Text | Query type corresponding to the statement. |
| except_info | Text | Information about the exception rule triggered by the statement. |
| unique_plan_id | Bigint | ID of the normalized unique plan. |
| sql_hash | Text | Normalized SQL hash. |
| plan_hash | Text | Normalized plan hash. |
| use_plan_baseline | Text | Indicates whether the bound plan is used for executing the current statement. If it is used, the name of the plan_baseline column in pg_plan_baseline is displayed. |

| Column | Type | Description |
|-----------------------------|--------------|---|
| outline_name | Text | Name of the outline used for the statement plan. |
| loader_status | Text | The JSON string for storing import and export service information is as follows.
<ol style="list-style-type: none">1. address: indicates the IP address of the peer cluster. The port number is displayed for the source cluster.2. direction: indicates the import and export service type. The value can be gds to file, gds from file, gds to pipe, gds from pipe, copy from or copy to.3. min/max/total_lines/bytes: indicates the minimum value, maximum value, total lines, and bytes of the import and export statements on all DNs. |
| parse_time | Bigint | Total parsing time before the statement is queued (including lexical and syntax parsing, optimization rewriting, and plan generation time), in milliseconds. This column is available only in clusters of version 8.3.0.100 or later. |
| disk_cache_hit_ratio | numeric(5,2) | Disk cache hit rate. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_read_size | Bigint | Total size of data read from disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_write_size | Bigint | Total size of data written to disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_size | Bigint | Total size of data read remotely from OBS due to disk cache read failure, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_time | Bigint | Total number of times data is read remotely from OBS due to disk cache read failure. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|---|
| vfs_scan_bytes | Bigint | Total number of bytes scanned by the OBS virtual file system in response to upper-layer requests, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| vfs_remote_read_bytes | Bigint | Total number of bytes actually read from OBS by the OBS virtual file system, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| preload_submit_time | Bigint | Total time for submitting I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_time | Bigint | Total time for waiting for I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_count | Bigint | Total number of times that the prefetching process waits for I/O requests. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| disk_cache_load_time | Bigint | Total time for reading from disk cache, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_conflict_count | Bigint | Number of times a block in the disk cache produces a hash conflict. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_error_count | Bigint | Number of disk cache read failures. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|---|
| disk_cache_error_code | Bigint | <p>Error code for disk cache read failures. Multiple error codes may be generated. If the disk cache fails to be read, OBS remote read is initiated and cache blocks are rewritten. The error code types are as follows: This column only applies to OBS 3.0 tables and foreign tables.</p> <ul style="list-style-type: none">• 1: A hash conflict occurs in the disk cache block.• 2: The generation time of the disk cache block is later than that of the OldestXmin transaction.• 4: Invoking the pread system when reading cache files from the disk cache failed.• 8: The data version of the disk cache block does not match.• 16: The version of the data written to the write cache does not match the latest version.• 32: Opening the cache file corresponding to the cache block failed.• 64: The size of the data read from the disk cache does not match.• 128: The CSN recorded in the disk cache block does not match. |
| obs_io_req_avg_rtt | Bigint | Average Round Trip Time (RTT) for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_avg_latency | Bigint | Average delay for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_1s | Bigint | Number of OBS I/O requests with a latency exceeding 1 second. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_10s | Bigint | Number of OBS I/O requests with a latency exceeding 10 seconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_count | Bigint | Total number of OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|-----------------------------|--------|---|
| obs_io_req_retry_count | Bigint | Total number of retries for OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_rate_limit_count | Bigint | Total number of times OBS I/O requests are flow-controlled. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

18.3.91 GS_WLM_SESSION_STATISTICS

GS_WLM_SESSION_STATISTICS displays load management information about jobs being executed by the current user on the current CN.

Table 18-163 GS_WLM_SESSION_STATISTICS columns

| Column | Type | Description |
|------------------|---------|--|
| datid | OID | OID of the database this backend is connected to |
| dbname | Name | Name of the database the backend is connected to |
| schemaname | Text | Schema name |
| nodename | Text | Name of the CN where the statement is executed |
| username | Name | User name used for connecting to the backend |
| application_name | Text | Name of the application that is connected to the backend |
| client_addr | inet | IP address of the client connected to this backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used |

| Column | Type | Description |
|---------------------|--------------------------|---|
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |
| pid | Bigint | Process ID of the backend |
| block_time | Bigint | Block time before the statement is executed. The unit is ms. |
| start_time | Timestamp with time zone | Time when the statement starts to be executed |
| duration | Bigint | For how long a statement has been executing. The unit is ms. |
| estimate_total_time | Bigint | Estimated execution time of a statement. The unit is ms. |
| estimate_left_time | Bigint | Estimated remaining time of statement execution. The unit is ms. |
| enqueue | Text | Workload management resource status |
| resource_pool | Name | Resource pool used by the user |
| control_group | Text | Cgroup used by the statement |
| estimate_memory | Integer | Estimated memory used by a statement on a single instance. The unit is MB. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DNs. The unit is MB. |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DNs. The unit is MB. |
| average_peak_memory | Integer | Average memory usage during statement execution. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of a statement among DNs. |
| spill_info | Text | Statement spill information on all DNs.
None : The statement has not been spilled to disks on any DNs.
All : The statement has been spilled to disks on all DNs.
[a:b] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DNs. |
| min_spill_size | Integer | Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 . |

| Column | Type | Description |
|---------------------|---------|---|
| max_spill_size | Integer | Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_dn_time | Bigint | Minimum execution time of a statement across all DNs. The unit is ms. |
| max_dn_time | Bigint | Maximum execution time of a statement across all DNs. The unit is ms. |
| average_dn_time | Bigint | Average execution time of a statement across all DNs. The unit is ms. |
| dntime_skew_percent | Integer | Execution time skew of a statement among DNs. |
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DNs. The unit is ms. |
| cpu_skew_percent | Integer | CPU time skew of a statement among DNs. |
| min_peak_iops | Integer | Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| max_peak_iops | Integer | Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| average_peak_iops | Integer | Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| iops_skew_percent | Integer | I/O skew across DNs. |

| Column | Type | Description |
|---------------------|---------|--|
| min_read_speed | Integer | Minimum I/O read rate of a statement across all DN's within a monitoring period (5s). The unit is KB/s. |
| max_read_speed | Integer | Maximum I/O read rate of a statement across all DN's within a monitoring period (5s). The unit is KB/s. |
| average_read_speed | Integer | Average I/O read rate of a statement across all DN's within a monitoring period (5s). The unit is KB/s. |
| min_write_speed | Integer | Minimum I/O write rate of a statement across all DN's within a monitoring period (5s). The unit is KB/s. |
| max_write_speed | Integer | Maximum I/O write rate of a statement across all DN's within a monitoring period (5s). The unit is KB/s. |
| average_write_speed | Integer | Average I/O write rate of a statement across all DN's within a monitoring period (5s). The unit is KB/s. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DN's. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DN's. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| warning | Text | Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed:
<ol style="list-style-type: none">1. The max spill size exceeds the alarm size 256MB2. The max broadcast size exceeds the alarm size 100MB3. Early spill4. Spill times is greater than 35. Spill on memory adaptive6. Hash table conflict7. NestLoop in hashjoin |
| unique_sql_id | Bigint | ID of the normalized unique SQL. |
| queryid | Bigint | Internal query ID used for statement execution |

| Column | Type | Description |
|-----------------------------|---------------|--|
| query | Text | Statement that is being executed |
| query_plan | Text | Execution plan of a statement
Specification restrictions:
1. Execution plans are displayed only for DML statements.
2. In 8.2.1.100 and later versions, the number of data binding times is added to the execution plans of Parse Bind Execute (PBE) statements to facilitate statement analysis. The number of data binding times is displayed in the format of PBE bind times: Times . |
| node_group | Text | Logical cluster of the user running the statement |
| stmt_type | Text | Query type corresponding to the statement. |
| except_info | Text | Information about the exception rule triggered by the statement. |
| parse_time | Bigint | Total parsing time before the statement is queued (including lexical and syntax parsing, optimization rewriting, and plan generation time), in milliseconds.

This column is only supported in version 8.3.0.100 or later. |
| unique_plan_id | Bigint | ID of the normalized unique plan. |
| sql_hash | Text | Normalized SQL hash. |
| plan_hash | Text | Normalized plan hash. |
| disk_cache_hit_ratio | numeric(5, 2) | Disk cache hit rate. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_disk_read_size | Bigint | Total size of data read from disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_disk_write_size | Bigint | Total size of data written to disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_remote_read_size | Bigint | Total size of data read remotely from OBS due to disk cache read failure, in MB. This column only applies to OBS 3.0 tables and foreign tables. |

| Column | Type | Description |
|-----------------------------|--------|---|
| disk_cache_remote_read_time | Bigint | Total number of times data is read remotely from OBS due to disk cache read failure. This column only applies to OBS 3.0 tables and foreign tables. |
| block_name | Text | Name of the interception rule that matches the statement. This column is available only in clusters of version 9.1.0.200 or later. |

18.3.92 GS_WLM_SQL_ALLOW

The **GS_WLM_SQL_ALLOW** view displays the configured resource management SQL whitelist.

The whitelist contains:

- Default SQL whitelist of the system.
- SQL whitelist specified by the GUC parameter **wlm_sql_allow_list**.

18.3.93 GS_WORKLOAD_SQL_COUNT

GS_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in workload Cgroups on the current node, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements.

Table 18-164 GS_WORKLOAD_SQL_COUNT columns

| Column | Type | Description |
|--------------|--------|------------------------------------|
| workload | Name | Workload Cgroup name |
| select_count | Bigint | Number of SELECT statements |
| update_count | Bigint | Number of UPDATE statements |
| insert_count | Bigint | Number of INSERT statements |
| delete_count | Bigint | Number of DELETE statements |
| ddl_count | Bigint | Number of DDL statements |
| dml_count | Bigint | Number of DML statements |

| Column | Type | Description |
|-----------|--------|---------------------------------|
| dcl_count | Bigint | Number of DCL statements |

18.3.94 GS_WORKLOAD_SQL_ELAPSE_TIME

GS_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in workload Cgroups on the current node, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond.

Table 18-165 GS_WORKLOAD_SQL_ELAPSE_TIME columns

| Column | Type | Description |
|---------------------|--------|---|
| workload | Name | Workload Cgroup name |
| total_select_elapse | Bigint | Total response time of SELECT statements |
| max_select_elapse | Bigint | Maximum response time of SELECT statements |
| min_select_elapse | Bigint | Minimum response time of SELECT statements |
| avg_select_elapse | Bigint | Average response time of SELECT statements |
| total_update_elapse | Bigint | Total response time of UPDATE statements |
| max_update_elapse | Bigint | Maximum response time of UPDATE statements |
| min_update_elapse | Bigint | Minimum response time of UPDATE statements |
| avg_update_elapse | Bigint | Average response time of UPDATE statements |
| total_insert_elapse | Bigint | Total response time of INSERT statements |
| max_insert_elapse | Bigint | Maximum response time of INSERT statements |
| min_insert_elapse | Bigint | Minimum response time of INSERT statements |
| avg_insert_elapse | Bigint | Average response time of INSERT statements |

| Column | Type | Description |
|---------------------|--------|---|
| total_delete_elapse | Bigint | Total response time of DELETE statements |
| max_delete_elapse | Bigint | Maximum response time of DELETE statements |
| min_delete_elapse | Bigint | Minimum response time of DELETE statements |
| avg_delete_elapse | Bigint | Average response time of DELETE statements |

18.3.95 GS_WORKLOAD_TRANSACTION

GS_WORKLOAD_TRANSACTION provides transaction information about workload cgroups on a single CN. The database records the number of times that each workload Cgroup commits and rolls back transactions and the response time of transaction commitment and rollback, in microseconds.

Table 18-166 GS_WORKLOAD_TRANSACTION columns

| Column | Type | Description |
|------------------|--------|-----------------------|
| workload | Name | Workload Cgroup name |
| commit_counter | Bigint | Number of the commits |
| rollback_counter | Bigint | Number of rollbacks |
| resp_min | Bigint | Minimum response time |
| resp_max | Bigint | Maximum response time |
| resp_avg | Bigint | Average response time |
| resp_total | Bigint | Total response time |

18.3.96 MPP_TABLES

MPP_TABLES displays information about tables in PGXC_CLASS.

Table 18-167 MPP_TABLES columns

| Name | Type | Description |
|------------|------|--|
| schemaname | name | Name of the schema that contains the table |
| tablename | name | Name of a table |

| Name | Type | Description |
|------------|------------------|--|
| tableowner | name | Owner of the table |
| tablespace | name | Tablespace where the table is located. |
| pggroup | name | Name of a node cluster. |
| nodeoids | oidvector_extend | List of distributed table node OIDs |

18.3.97 PG_AVAILABLE_EXTENSION_VERSIONS

PG_AVAILABLE_EXTENSION_VERSIONS displays the extension versions of certain database features.

Table 18-168 PG_AVAILABLE_EXTENSION_VERSIONS columns

| Column | Type | Description |
|-------------|---------|--|
| Name | Name | Extension name |
| version | Text | Version name |
| installed | boolean | The value is true if the version of this extension is currently installed. |
| superuser | boolean | The value is true if only system administrators are allowed to install this extension. |
| relocatable | boolean | The value is true if an extension can be relocated to another schema. |
| schema | Name | Name of the schema that the extension must be installed into. The value is NULL if the extension is partially or fully relocatable. |
| requires | name[] | Names of prerequisite extensions. The value is NULL if there are no prerequisite extensions. |
| comment | Text | Comment string from the extension's control file |

18.3.98 PG_AVAILABLE_EXTENSIONS

PG_AVAILABLE_EXTENSIONS displays the extended information about certain database features.

Table 18-169 PG_AVAILABLE_EXTENSIONS columns

| Column | Type | Description |
|-------------------|------|--|
| Name | Name | Extension name. |
| default_version | Text | Name of default version. The value is NULL if none is specified. |
| installed_version | Text | Currently installed version of the extension. The value is NULL if no version is installed. |
| comment | Text | Comment string from the extension's control file. |

18.3.99 PG_BULKLOAD_STATISTICS

On any normal node in a cluster, **PG_BULKLOAD_STATISTICS** displays the execution status of the import and export services. Each import or export service corresponds to a record. This view is accessible only to users with system administrators rights.

Table 18-170 PG_BULKLOAD_STATISTICS columns

| Name | Type | Description |
|-------------|--------------------------|---|
| node_name | Text | Node name |
| db_name | Text | Database name |
| query_id | Bigint | Query ID. It is equivalent to debug_query_id . |
| tid | Bigint | ID of the current thread |
| lwtid | Integer | Lightweight thread ID |
| session_id | Bigint | GDS session ID |
| direction | Text | Service type. The options are gds to file , gds from file , gds to pipe , gds from pipe , copy from , and copy to . |
| query | Text | Query statement |
| address | Text | Location of the foreign table used for data import and export |
| query_start | Timestamp with time zone | Start time of data import or export |

| Name | Type | Description |
|-------------|--------|--|
| total_bytes | Bigint | Total size of data to be processed
This parameter is specified only when a GDS common file is to be imported and the record in the row comes from a CN. Otherwise, left this parameter unspecified. |
| phase | Text | Execution phase of the current service import and export. The options are INITIALIZING , TRANSFER_DATA , and RELEASE_RESOURCE . |
| done_lines | Bigint | Number of lines that have been transferred |
| done_bytes | Bigint | Number of bytes that have been transferred |

18.3.100 PG_COMM_CLIENT_INFO

PG_COMM_CLIENT_INFO stores the client connection information of a single node. (You can query this view on a DN to view the information about the connection between the CN and DN.)

Table 18-171 PG_COMM_CLIENT_INFO columns

| Column | Type | Description |
|-------------|---------|---|
| node_name | Text | Current node name. |
| app | Text | Client application name |
| tid | Bigint | Thread ID of the current thread. |
| lwtid | Integer | Lightweight thread ID of the current thread. |
| query_id | Bigint | Query ID. It is equivalent to debug_query_id . |
| socket | Integer | It is displayed if the connection is a physical connection. |
| remote_ip | Text | Peer node IP address. |
| remote_port | Text | Peer node port. |
| logic_id | Integer | If the connection is a logical connection, sid is displayed. If -1 is displayed, the current connection is a physical connection. |

18.3.101 PG_COMM_DELAY

PG_COMM_DELAY displays the communication library delay status for a single DN.

Table 18-172 PG_COMM_DELAY columns

| Column | Type | Description |
|-------------|---------|---|
| node_name | Text | Node name |
| remote_name | Text | Name of the node with the maximum latency in connecting to the peer end. |
| remote_host | Text | IP address of the peer. |
| stream_num | Integer | Number of logical stream connections used by the current physical connection. |
| min_delay | Integer | Minimum delay of the current physical connection. The unit is microsecond. |
| average | Integer | Average delay of the current physical connection. The unit is microsecond. |
| max_delay | Integer | Maximum delay of the current physical connection. The unit is microsecond.
NOTE
If its value is -1, the latency detection has timed out. In this case, re-establish the connection between nodes and then perform the query. |

18.3.102 PG_COMM_STATUS

PG_COMM_STATUS displays the communication library status for a single DN.

Table 18-173 PG_COMM_STATUS columns

| Column | Type | Description |
|-----------|---------|--|
| node_name | Text | Node name. |
| rxpck/s | Integer | Receiving rate of the communication library on a node. The unit is byte/s. |
| txpck/s | Integer | Sending rate of the communication library on a node. The unit is byte/s. |
| rxkB/s | Bigint | Receiving rate of the communication library on a node. The unit is KB/s. |
| txkB/s | Bigint | Sending rate of the communication library on a node. The unit is KB/s. |

| Column | Type | Description |
|----------------|---------|--|
| buffer | Bigint | Size of the buffer of the Cmailbox. |
| memKB(libcomm) | Bigint | Communication memory size of the libcomm process, in KB. |
| memKB(libpq) | Bigint | Communication memory size of the libpq process, in KB. |
| %USED(PM) | Integer | Real-time usage of the postmaster thread. |
| %USED (sflow) | Integer | Real-time usage of the gs_sender_flow_controller thread. |
| %USED (rflow) | Integer | Real-time usage of the gs_receiver_flow_controller thread. |
| %USED (rloop) | Integer | Highest real-time usage among multiple gs_receivers_loop threads. |
| stream | Integer | Total number of used logical connections. |

18.3.103 PG_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of all the communication libraries for a single DN.

Table 18-174 PG_COMM_RECV_STREAM columns

| Column | Type | Description |
|-------------|---------|--------------------------------------|
| node_name | Text | Node name |
| local_tid | Bigint | ID of the thread using this stream |
| remote_name | Text | Name of the peer node |
| remote_tid | Bigint | Peer thread ID |
| idx | Integer | Peer DN ID in the local DN |
| sid | Integer | Stream ID in the physical connection |
| tcp_sock | Integer | TCP socket used in the stream |

| Column | Type | Description |
|-------------|---------|---|
| state | Text | Current status of the stream <ul style="list-style-type: none">• UNKNOWN: The logical connection is unknown.• READY: The logical connection is ready.• RUN: The logical connection receives packets normally.• HOLD: The logical connection is waiting to receive packets.• CLOSED: The logical connection is closed.• TO_CLOSED: The logical connection is to be closed.• WRITING: Data is being written. |
| query_id | Bigint | debug_query_id corresponding to the stream |
| pn_id | Integer | plan_node_id of the query executed by the stream |
| send_smp | Integer | smpid of the sender of the query executed by the stream |
| recv_smp | Integer | smpid of the receiver of the query executed by the stream |
| recv_bytes | Bigint | Total data volume received from the stream. The unit is byte. |
| time | Bigint | Current life cycle service duration of the stream. The unit is ms. |
| speed | Bigint | Average receiving rate of the stream. The unit is byte/s. |
| quota | Bigint | Current communication quota value of the stream. The unit is Byte. |
| buff_useize | Bigint | Current size of the data cache of the stream. The unit is byte. |

18.3.104 PG_COMM_SEND_STREAM

PG_COMM_SEND_STREAM displays the sending stream status of all the communication libraries for a single DN.

Table 18-175 PG_COMM_SEND_STREAM columns

| Column | Type | Description |
|-----------|------|-------------|
| node_name | Text | Node name |

| Column | Type | Description |
|-------------|---------|---|
| local_tid | Bigint | ID of the thread using this stream |
| remote_name | Text | Name of the peer node |
| remote_tid | Bigint | Peer thread ID |
| idx | Integer | Peer DN ID in the local DN |
| sid | Integer | Stream ID in the physical connection |
| tcp_sock | Integer | TCP socket used in the stream |
| state | Text | Current status of the stream <ul style="list-style-type: none">• UNKNOWN: The logical connection is unknown.• READY: The logical connection is ready.• RUN: The logical connection sends packets normally.• HOLD: The logical connection is waiting to send packets.• CLOSED: The logical connection is closed.• TO_CLOSED: The logical connection is to be closed.• WRITING: Data is being written. |
| query_id | Bigint | debug_query_id corresponding to the stream |
| pn_id | Integer | plan_node_id of the query executed by the stream |
| send_smp | Integer | smpid of the sender of the query executed by the stream |
| recv_smp | Integer | smpid of the receiver of the query executed by the stream |
| send_bytes | Bigint | Total data volume sent by the stream. The unit is Byte. |
| time | Bigint | Current life cycle service duration of the stream. The unit is ms. |
| speed | Bigint | Average sending rate of the stream. The unit is Byte/s. |
| quota | Bigint | Current communication quota value of the stream. The unit is Byte. |
| wait_quota | Bigint | Extra time generated when the stream waits the quota value. The unit is ms. |

18.3.105 PG_COMM_QUERY_SPEED

PG_COMM_QUERY_SPEED displays traffic information about all queries on a single node.

Table 18-176 PG_COMM_QUERY_SPEED columns

| Column | Type | Description |
|-----------|--------|--|
| node_name | Text | Node name |
| query_id | Bigint | debug_query_id corresponding to the stream |
| rxkB/s | Bigint | Receiving rate of the query stream (unit: byte/s) |
| txkB/s | Bigint | Sending rate of the query stream (unit: byte/s) |
| rxkB | Bigint | Total received data of the query stream (unit: byte) |
| txkB | Bigint | Total sent data of the query stream (unit: byte) |
| rxpck/s | Bigint | Packet receiving rate of the query (unit: packets/s) |
| txpck/s | Bigint | Packet sending rate of the query (unit: packets/s) |
| rxpck | Bigint | Total number of received packets of the query |
| txpck | Bigint | Total number of sent packets of the query |

18.3.106 PG_CONTROL_GROUP_CONFIG

PG_CONTROL_GROUP_CONFIG displays the Cgroup configuration information in the system.

Table 18-177 PG_CONTROL_GROUP_CONFIG columns

| Column | Type | Description |
|-------------------------|------|---|
| pg_control_group_config | Text | Configuration information of the Cgroup |

18.3.107 PG_CURSORS

PG_CURSORS displays available cursors.

Table 18-178 PG_CURSORS columns

| Column | Type | Description |
|---------------|--------------------------|---|
| Name | Text | Cursor name. |
| statement | Text | Query statement when the cursor is declared to change |
| is_holdable | boolean | Whether the cursor is holdable (that is, it can be accessed after the transaction that declared the cursor has committed). If it is, its value is true . |
| is_binary | boolean | Whether the cursor was declared BINARY. If it was, its value is true . |
| is_scrollable | boolean | Whether the cursor is scrollable (that is, it allows rows to be retrieved in a nonsequential manner). If it is, its value is true . |
| creation_time | Timestamp with time zone | Timestamp of the cursor. |

18.3.108 PG_EXT_STATS

PG_EXT_STATS displays extension statistics stored in the [PG_STATISTIC_EXT](#) table. The extension statistics means multiple columns of statistics.

Table 18-179 PG_EXT_STATS columns

| Column | Type | Reference | Description |
|------------|------------|---|--|
| schemaname | Name | PG_NAMESPACE.nspname | Name of the schema that contains a table |
| tablename | Name | PG_CLASS.relname | Name of a table |
| attname | int2vector | PG_STATISTIC_EXT.stakey | Indicates the columns to be combined for collecting statistics. |
| inherited | boolean | - | Includes inherited sub-columns if the value is true ; otherwise, indicates the column in a specified table. |
| null_frac | Real | - | Percentage of column combinations that are null to all records |
| avg_width | Integer | - | Average width of column combinations. The unit is byte. |

| Column | Type | Reference | Description |
|------------------|----------|-----------|--|
| n_distinct | Real | - | <ul style="list-style-type: none">Estimated number of distinct values in a column combination if the value is greater than 0Negative of the number of distinct values divided by the number of rows if the value is less than 0The number of distinct values is unknown if the value is 0. NOTE
The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows.
The positive form is used when the column seems to have a fixed number of possible values. For example, -1 indicates that the number of distinct values is the same as the number of rows for a column combination. |
| n_dndistinct | Real | - | <p>Number of unique not-null data values in the dn1 column combination</p> <ul style="list-style-type: none">Exact number of distinct values if the value is greater than 0Negative of the number of distinct values divided by the number of rows if the value is less than 0 For example, if a value in a column combination appears twice in average, n_dndistinct equals -0.5.The number of distinct values is unknown if the value is 0. |
| most_common_vals | anyarray | - | <p>List of the most common values in a column combination. If this combination does not have the most common values, most_common_vals_null will be NULL. None of the most common values in most_common_vals is NULL.</p> |

| Column | Type | Reference | Description |
|------------------------|----------|-----------|---|
| most_common_freqs | real[] | - | List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals is NULL) |
| most_common_vals_null | anyarray | - | List of the most common values in a column combination. If this combination does not have the most common values, most_common_vals_null will be NULL. At least one of the common values in most_common_vals_null is NULL. |
| most_common_freqs_null | real[] | - | List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals_null is NULL) |

18.3.109 PG_GET_INVALID_BACKENDS

PG_GET_INVALID_BACKENDS displays the information about backend threads on the CN that are connected to the current standby DN.

Table 18-180 PG_GET_INVALID_BACKENDS columns

| Column | Type | Description |
|---------------|--------------------------|--|
| pid | Bigint | Thread ID |
| node_name | Text | Node information connected to the backend thread |
| dbname | Name | Name of the connected database |
| backend_start | Timestamp with time zone | Backend thread startup time |
| query | Text | Query statement performed by the backend thread |

18.3.110 PG_GET_SENDERS_CATCHUP_TIME

PG_GET_SENDERS_CATCHUP_TIME displays the catchup information of the currently active primary/standby instance sending thread on a single DN.

Table 18-181 PG_GET_SENDERS_CATCHUP_TIME columns

| Column | Type | Description |
|------------------------|--------------------------|--|
| pid | Bigint | Current sender thread ID |
| lwpid | Integer | Current sender lwpid |
| local_role | Text | Local role |
| peer_role | Text | Peer role |
| state | Text | Current sender's replication status |
| type | Text | Current sender type |
| catchup_start | Timestamp with time zone | Startup time of a catchup task |
| catchup_end | Timestamp with time zone | End time of a catchup task |
| catchup_type | Text | Catchup task type, full or incremental |
| catchup_bcm_filename | Text | BCM file executed by the current catchup task |
| catchup_bcm_finished | Integer | Number of BCM files completed by a catchup task |
| catchup_bcm_total | Integer | Total number of BCM files to be operated by a catchup task |
| catchup_percent | Text | Completion percentage of a catchup task |
| catchup_remaining_time | Text | Estimated remaining time of a catchup task |

18.3.111 PG_GLOBAL_TEMP_ATTACHED_PIDS

This view displays information about sessions of resources occupied by global temporary tables on the current node. This view is supported only by clusters of 8.2.1.220 and later versions.

Table 18-182 PG_GLOBAL_TEMP_ATTACHED_PIDS columns

| Column | Type | Description |
|------------|------|-------------|
| schemaname | Name | Schema name |

| Column | Type | Description |
|-----------|--------|------------------|
| tablename | Name | Table name |
| pid | Bigint | PID of a session |

18.3.112 PG_GROUP

PG_GROUP displays the database role authentication and the relationship between roles.

Table 18-183 PG_GROUP columns

| Column | Type | Description |
|----------|-------|--|
| groname | Name | Group name |
| grosysid | OID | Group ID |
| grolist | oid[] | An array, including all the role IDs in this group |

18.3.113 PG_INDEXES

PG_INDEXES displays access to useful information about each index in the database.

Table 18-184 PG_INDEXES columns

| Column | Type | Reference | Description |
|------------|------|---------------------------------------|--|
| schemaname | Name | PG_NAMESPACE.nspname | Name of the schema that contains tables and indexes |
| tablename | Name | PG_CLASS.relname | Name of the table for which the index serves |
| indexname | Name | PG_CLASS.relname | Index name |
| tablespace | Name | PG_TABLESPACE.spcname | Name of the tablespace that contains the index |
| indexdef | Text | N/A | Index definition (a reconstructed CREATE INDEX command) |

Example

Query the index information about a specified table.

```
SELECT * FROM pg_indexes WHERE tablename = 'mytable';
schemaname | tablename | indexname | tablespace | indexdef
```

```
-----+-----+-----+-----+
+-----+-----+-----+-----+
public | mytable | idx_mytable_id | | CREATE INDEX idx_mytable_id ON mytable USING btree
(id) TABLESPACE pg_default
(1 row)
```

Query information about indexes of all tables in a specified schema in the current database.

```
SELECT tablename, indexname, indexdef FROM pg_indexes WHERE schemaname = 'public' ORDER BY
tablename,indexname;
tablename | indexname | indexdef
-----+-----+-----+
books | books_pkey | CREATE UNIQUE INDEX books_pkey ON books USING btree (id) TABLESPACE
pg_default
books | idx_books_tags_gin | CREATE INDEX idx_books_tags_gin ON books USING gin (tags)
TABLESPACE pg_default
customer | c_custkey_key | CREATE UNIQUE INDEX c_custkey_key ON customer USING btree
(c_custkey, c_name) TABLESPACE pg_default
mytable | idx_mytable_id | CREATE INDEX idx_mytable_id ON mytable USING btree (id) TABLESPACE
pg_default
test1 | idx_test_id | CREATE INDEX idx_test_id ON test1 USING btree (id) TABLESPACE pg_default
v0 | v0_pkey | CREATE UNIQUE INDEX v0_pkey ON v0 USING btree (c) TABLESPACE pg_default
(6 rows)
```

18.3.114 PG_JOB

PG_JOB displays detailed information about scheduled tasks created by users.

The **PG_JOB** view replaces the **PG_JOB** system catalog in earlier versions and provides forward compatibility with earlier versions. The original **PG_JOB** system catalog is changed to the **PG_JOBS** system catalog. For details about **PG_JOBS**, see [PG_JOBS](#).

Table 18-185 PG_JOB columns

| Column | Type | Description |
|----------------------|--------|---|
| job_id | Bigint | Job ID |
| current_postgres_pid | Bigint | If the current job has been executed, the PostgreSQL thread ID of this job is recorded. The default value is -1 , indicating that the task is not executed or has been executed. |
| log_user | Name | User name of the job creator |
| priv_user | Name | User name of the job executor |
| dbname | Name | Name of the database where the job is executed |
| node_name | Name | CN node on which the job will be created and executed |

| Column | Type | Description |
|-----------------|-----------------------------|--|
| job_status | Text | <p>Status of the current job. The value range is r, s, f, d, p, w, or l. The default value is s. The indications are as follows:</p> <ul style="list-style-type: none">• r=running• s=successfully finished• f=job failed• d=disable• p=pending• w=waiting• l=launching <p>NOTE</p> <ul style="list-style-type: none">• Note: When you disable a scheduled task (by setting job_queue_processes to 0), the thread monitor the job execution is not started, and the job_status will not be updated. You can ignore the job_status.• Only when the scheduled task function is enabled (that is, when job_queue_processes is not 0), the system updates the value of job_status based on the real-time job status. |
| start_date | Timestamp without time zone | Start time of the first job execution, precise to millisecond |
| next_run_date | Timestamp without time zone | Scheduled time of the next job execution, accurate to millisecond |
| failure_count | Smallint | Number of consecutive failures |
| interval | Text | Job execution interval |
| last_start_date | Timestamp without time zone | Start time of the last job execution, accurate to millisecond |
| last_end_date | Timestamp without time zone | End time of the last job execution, accurate to millisecond |
| last_suc_date | Timestamp without time zone | Start time of the last successful job execution, accurate to millisecond |
| this_run_date | Timestamp without time zone | Start time of the ongoing job execution, accurate to millisecond |
| nspname | Name | Name of the namespace where a job is running |

| Column | Type | Description |
|--------|------|-------------|
| what | Text | Job content |

18.3.115 PG_JOB_PROC

The **PG_JOB_PROC** view replaces the **PG_JOB_PROC** system catalog in earlier versions and provides forward compatibility with earlier versions. The original **PG_JOB_PROC** and **PG_JOB** system catalogs are merged into the **PG_JOBS** system catalog in the current version. For details about the **PG_JOBS** system catalog, see [PG_JOBS](#).

Table 18-186 PG_JOB_PROC columns

| Column | Type | Description |
|--------|--------|-------------|
| job_id | Bigint | Job ID |
| what | Text | Job content |

18.3.116 PG_JOB_SINGLE

PG_JOB_SINGLE displays job information about the current node.

Table 18-187 PG_JOB_SINGLE columns

| Column | Type | Description |
|----------------------|--------|---|
| job_id | Bigint | Job ID |
| current_postgres_pid | Bigint | If the current job has been executed, the PostgreSQL thread ID of this job is recorded. The default value is -1, indicating that the task is not executed or has been executed. |
| log_user | Name | User name of the job creator |
| priv_user | Name | User name of the job executor |
| dbname | Name | Name of the database where the job is executed |
| node_name | Name | CN node on which the job will be created and executed |

| Column | Type | Description |
|-----------------|-----------------------------|--|
| job_status | Text | <p>Status of the current job. The value range is r, s, f, d, p, w, or l. The default value is s. The indications are as follows:</p> <ul style="list-style-type: none">• r=running• s=successfully finished• f=job failed• d=disable• p=pending• w=waiting• l=launching <p>NOTE</p> <ul style="list-style-type: none">• Note: When you disable a scheduled task (by setting job_queue_processes to 0), the thread monitor the job execution is not started, and the job_status will not be updated. You can ignore the job_status.• Only when the scheduled task function is enabled (that is, when job_queue_processes is not 0), the system updates the value of job_status based on the real-time job status. |
| start_date | Timestamp without time zone | Start time of the first job execution, precise to millisecond |
| next_run_date | Timestamp without time zone | Scheduled time of the next job execution, accurate to millisecond |
| failure_count | Smallint | Number of consecutive failures. |
| interval | Text | Job execution interval |
| last_start_date | Timestamp without time zone | Start time of the last job execution, accurate to millisecond |
| last_end_date | Timestamp without time zone | End time of the last job execution, accurate to millisecond |
| last_suc_date | Timestamp without time zone | Start time of the last successful job execution, accurate to millisecond |
| this_run_date | Timestamp without time zone | Start time of the ongoing job execution, accurate to millisecond |
| nspname | Name | Name of the namespace where a job is running |

| Column | Type | Description |
|--------|------|-------------|
| what | Text | Job content |

18.3.117 PG_LIFECYCLE_DATA_DISTRIBUTE

PG_LIFECYCLE_DATA_DISTRIBUTE displays the distribution of cold and hot data in a multi-temperature table of OBS.

Table 18-188 PG_LIFECYCLE_DATA_DISTRIBUTE columns

| Column | Type | Description |
|---------------------|------|---|
| schemaname | Name | Schema name |
| tablename | Name | Current table name |
| nodename | Name | Node name |
| hotpartition | Text | Hot partition on the DN |
| coldpartition | Text | Cold partition on the DN |
| switchablepartition | Text | Switchable partition on the DN |
| hotdatasize | Text | Data size of the hot partition on the DN |
| colddatasize | Text | Data size of the cold partition on the DN |
| switchabledatasize | Text | Data size of the switchable partition on the DN |

18.3.118 PG_LOCKS

PG_LOCKS displays information about the locks held by open transactions.

Table 18-189 PG_LOCKS columns

| Column | Type | Reference | Description |
|----------|------|-----------|---|
| locktype | Text | N/A | Type of the locked object: relation, extend, page, tuple, transactionid, virtualxid, object, userlock, and advisory |

| Column | Type | Reference | Description |
|--------------------|----------|---------------------------------|---|
| database | OID | PG_DATABASE.oid | OID of the database in which the locked target exists <ul style="list-style-type: none">The OID is 0 if the target is a shared object.The OID is NULL if the locked target is a transaction. |
| relation | OID | PG_CLASS.oid | OID of the relationship targeted by the lock. The value is NULL if the object is neither a relationship nor part of a relationship. |
| page | Integer | N/A | Page number targeted by the lock within the relationship. If the object is neither a relation page nor row page, the value is NULL . |
| tuple | Smallint | N/A | Row number targeted by the lock within the page. If the object is not a row, the value is NULL . |
| virtualxid | Text | N/A | Virtual ID of the transaction targeted by the lock. If the object is not a virtual transaction ID, the value is NULL . |
| transactionid | Xid | N/A | ID of the transaction targeted by the lock. If the object is not a transaction ID, the value is NULL . |
| classid | OID | PG_CLASS.oid | OID of the system table that contains the object. If the object is not a general database object, the value is NULL . |
| objid | OID | N/A | OID of the lock target within its system table. If the target is not a general database object, the value is NULL . |
| objsubid | Smallint | N/A | Column number for a column in the table. The value is 0 if the target is some other object type. If the object is not a general database object, the value is NULL . |
| virtualtransaction | Text | N/A | Virtual ID of the transaction holding or awaiting this lock |

| Column | Type | Reference | Description |
|----------|--------------------------|-----------|--|
| pid | Bigint | N/A | Logical ID of the server thread holding or awaiting this lock. This is NULL if the lock is held by a prepared transaction. |
| mode | Text | N/A | Lock mode held or desired by this thread For more information about lock modes, see "LOCK" in <i>Data Warehouse Service (DWS) SQL Syntax Reference</i> . |
| granted | boolean | N/A | <ul style="list-style-type: none">The value is true if the lock is a held lock.The value is false if the lock is an awaited lock. |
| fastpath | boolean | N/A | Whether the lock is obtained through fast-path (true) or main lock table (false) |
| waittime | Timestamp with time zone | N/A | Timestamp when the lock wait starts.
This column is available only in clusters of version 9.1.0.200 or later. |
| holdtime | Timestamp with time zone | N/A | Timestamp when the lock starts to be held. This column is available only in clusters of version 9.1.0.200 or later. |

18.3.119 PG_LWLOCKS

PG_LWLOCKS provides information on lightweight locks currently held or being waited for by the current instance. This view is supported only by 9.1.0.200 and later cluster versions.

Table 18-190 PG_LWLOCKS columns

| Column | Type | Description |
|-----------|---------|---|
| pid | Bigint | ID of the backend thread. |
| query_id | Bigint | ID of a query. |
| lwtid | Integer | Lightweight thread ID of the backend thread. |
| reqlockid | Integer | ID of the lightweight lock that is being requested by the current thread. |

| Column | Type | Description |
|--------------|---------|---|
| reqlock | Text | Name of the lightweight lock corresponding to reqlockid . |
| heldlocknums | Integer | Number of lightweight locks obtained by the current thread. |
| heldlockid | Integer | Lightweight lock ID obtained by the current thread. |
| heldlock | Text | Name of the lightweight lock corresponding to heldlockid . |
| heldlockmode | Text | Lightweight lock mode corresponding to heldlockid . |

Example

Use the **PG_LWLOCKS** view to query information about lightweight locks that are being held or waiting for the current instance.

```
SELECT * FROM pg_lwlocks;
```

| | | | | | | | | |
|---|-------------------|-------|-----------|---------|--------------|------------|--------------|--------------|
| pid | query_id | lwtid | reqlockid | reqlock | heldlocknums | heldlockid | heldlock | heldlockmode |
| -----+-----+-----+-----+-----+-----+-----+----- | | | | | | | | |
| +-----+ | | | | | | | | |
| 139810224192480 | | 0 | 54842 | | 1 | 7 | WALWriteLock | Exclusive |
| 139810224199520 | 78250043526306022 | 54963 | | | 1 | 193860 | | |
| BUFFER_POOL_LWLOCK Exclusive | | | | | | | | |
| (2 rows) | | | | | | | | |

18.3.120 PG_NODE_ENV

PG_NODE_ENV displays the environmental variable information about the current node.

Table 18-191 PG_NODE_ENV columns

| Column | Type | Description |
|---------------|---------|--|
| node_name | Text | Name of the node |
| host | Text | Host name of the node |
| process | Integer | Number of the node process |
| port | Integer | Port ID of the node |
| installpath | Text | Installation directory of current node |
| datapath | Text | Data directory of the node |
| log_directory | Text | Log directory of the node |

18.3.121 PG_OS_THREADS

PG_OS_THREADS displays the status information about all the threads under the current node.

Table 18-192 PG_OS_THREADS columns

| Column | Type | Description |
|---------------|--------------------------|--|
| node_name | Text | Name of the node |
| pid | Bigint | Thread number running under the current node process |
| lwpid | Integer | Lightweight thread IDs corresponding to the PIDs |
| thread_name | Text | Thread names corresponding to the PIDs |
| creation_time | Timestamp with time zone | Creation time of the threads corresponding to the PIDs |

18.3.122 PG_POOLER_STATUS

PG_POOLER_STATUS displays the cache connection status in the pooler.
PG_POOLER_STATUS can only query on the CN, and displays the connection cache information about the pooler module.

Table 18-193 PG_POOLER_STATUS columns

| Column | Type | Description |
|------------|---------|---|
| database | Text | Database name |
| user_name | Text | Username |
| tid | Bigint | ID of the thread used for the connection to the CN |
| node_oid | Bigint | OID of the node connected |
| node_name | Name | Name of the node connected |
| in_use | boolean | Whether the connection is in use. The options are: <ul style="list-style-type: none">• t (true): The connection is in use.• f (false): The connection is not in use. |
| fdsock | Bigint | Peer socket |
| remote_pid | Bigint | Peer thread ID |

| Column | Type | Description |
|----------------|------|---|
| session_params | Text | GUC session parameter delivered by the connection |

Example

View information about the connection pool **pooler**:

```
select database,user_name,node_name,in_use,count(*) from pg_pooler_status group by 1, 2, 3,4 order by 5
desc limit 50;
database | user_name | node_name | in_use | count
-----+-----+-----+-----+-----
mydbdemo | user3     | cn_5001   | f      | 2
mydbdemo | user3     | dn_6005_6006 | t      | 2
mydbdemo | user3     | dn_6001_6002 | t      | 2
mydbdemo | user3     | dn_6003_6004 | f      | 2
mydbdemo | user3     | dn_6003_6004 | t      | 2
mydbdemo | user3     | dn_6005_6006 | f      | 2
mydbdemo | user3     | dn_6001_6002 | f      | 2
mydbdemo | user3     | cn_5002   | f      | 2
gaussdb  | user3     | dn_6003_6004 | f      | 1
mydbdemo | user3     | cn_5001   | t      | 1
music    | user2     | dn_6003_6004 | f      | 1
music    | user2     | dn_6005_6006 | f      | 1
gaussdb  | user1     | dn_6005_6006 | f      | 1
(13 rows)
```

18.3.123 PG_PREPARED_STATEMENTS

PG_PREPARED_STATEMENTS displays all prepared statements that are available in the current session.

Table 18-194 PG_PREPARED_STATEMENTS columns

| Column | Type | Description |
|-----------------|--------------------------|---|
| Name | Text | Identifier of a prepared statement. |
| statement | Text | Query string for creating this prepared statement. For prepared statements created through SQL, this is the PREPARE statement submitted by the client. For prepared statements created through the frontend/backend protocol, this is the text of the prepared statement itself. |
| prepare_time | Timestamp with time zone | Timestamp when the prepared statement is created. |
| parameter_types | regtype[] | Expected parameter types for the prepared statement in the form of an array of regtype . The OID corresponding to an element of this array can be obtained by casting the regtype value to oid. |

| Column | Type | Description |
|----------|---------|---|
| from_sql | boolean | <ul style="list-style-type: none">The value is true if the prepared statement is created using the PREPARE statement.The value is false if the instance is created using the frontend/backend protocol. |

18.3.124 PG_PREPARED_XACTS

PG_PREPARED_XACTS displays information about transactions that are currently prepared for two-phase commit.

Table 18-195 PG_PREPARED_XACTS columns

| Column | Type | Reference | Description |
|-------------|--------------------------|-------------------------------------|--|
| transaction | Xid | N/A | Numeric transaction identifier of the prepared transaction |
| gid | Text | N/A | Global transaction identifier that was assigned to the transaction |
| prepared | Timestamp with time zone | N/A | Time at which the transaction is prepared for commit |
| owner | Name | PG_AUTHID .rolname | Name of the user that executes the transaction |
| database | Name | PG_DATABASE .dbname | Name of the database in which the transaction is executed |

18.3.125 PG_PUBLICATION_TABLES

PG_PUBLICATION_TABLES displays the mapping between a publication and its published tables. Unlike the underlying system catalog [PG_PUBLICATION_REL](#), this view expands the publications defined as **FOR ALL TABLES** and **FOR ALL TABLES IN SCHEMA**, in which each publishable table has a row. This view is supported only by clusters of version 8.2.0.100 or later.

Table 18-196 PG_PUBLICATION_TABLES columns

| Column | Type | Description |
|---------|------|------------------|
| pubname | Name | Publication name |

| Column | Type | Description |
|------------|------|-------------------------------|
| schemaname | Name | Name of the schema of a table |
| tablename | Name | Table name |

Examples

Query all published tables.

```
SELECT * FROM PG_PUBLICATION_TABLES;  
pubname | schemaname | tablename  
-----+-----+-----  
mypub   | public    | t1  
mypub   | public    | t2  
(2 rows)
```

18.3.126 PG_QUERYBAND_ACTION

PG_QUERYBAND_ACTION displays information about the object associated with **query_band** and the **query_band** query order.

Table 18-197 PG_QUERYBAND_ACTION columns

| Column | Type | Description |
|------------|---------|---|
| qband | Text | query_band key-value pairs |
| respool_id | OID | OID of the resource pool associated with query_band |
| respool | Text | Name of the resource pool associated with query_band |
| priority | Text | Intra-queue priority associated with query_band |
| qborder | Integer | query_band query order |

18.3.127 PG_REPLICATION_SLOTS

PG_REPLICATION_SLOTS displays the replication node information.

Table 18-198 PG_REPLICATION_SLOTS columns

| Column | Type | Description |
|-----------|------|--|
| slot_name | Text | Name of a replication node |
| plugin | Name | Name of the output plug-in of the logical replication slot |
| slot_type | Text | Type of a replication node |

| Column | Type | Description |
|---------------|---------|--|
| datoid | OID | OID of the database on the replication node |
| database | Name | Name of the database on the replication node |
| active | boolean | Whether the replication node is active |
| xmin | Xid | Transaction ID of the replication node |
| catalog_xmin | Text | ID of the earliest-decoded transaction corresponding to the logical replication slot |
| restart_lsn | Text | Xlog file information on the replication node |
| dummy_standby | boolean | Whether the replication node is the dummy standby node |

18.3.128 PG_ROLES

PG_ROLES displays information about database roles.

Table 18-199 PG_ROLES columns

| Column | Type | Reference | Description |
|----------------|---------|-----------|--|
| rolname | Name | N/A | Role name |
| rolsuper | boolean | N/A | Whether the role is the initial system administrator with the highest permission |
| rolinherit | boolean | N/A | Whether the role inherits permissions for this type of roles |
| rolcreatorole | boolean | N/A | Whether the role can create other roles |
| rolcreatedb | boolean | N/A | Whether the role can create databases |
| rolcatupdate | boolean | N/A | Whether the role can update system tables directly. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users. |
| rolcanlogin | boolean | N/A | Whether the role can log in to the database |
| rolreplication | boolean | N/A | Whether the role can be replicated |
| rolauditadmin | boolean | N/A | Whether the role is an audit system administrator |

| Column | Type | Reference | Description |
|----------------|--------------------------|---------------------------------------|---|
| rolsystemadmin | boolean | N/A | Whether the role is a system administrator |
| rolconlimit | Integer | N/A | Limits the maximum number of concurrent connections of a user on a CN. -1 indicates no limit. |
| rolpassword | Text | N/A | Not the password (always reads as *****) |
| rolvalidbegin | Timestamp with time zone | N/A | Account validity start time; null if no start time |
| rolvaliduntil | Timestamp with time zone | N/A | Password expiry time; null if no expiration |
| rolrespool | Name | N/A | Resource pool that a user can use |
| rolparentid | OID | PG_AUTHID.rolparentid | OID of a group user to which the user belongs |
| roltabspace | Text | N/A | The storage space of the user permanent table. |
| roltemp space | Text | N/A | The storage space of the user temporary table. |
| rolspillspace | Text | N/A | The operator disk flushing space of the user. |
| rolconfig | Text[] | N/A | Session defaults for runtime configuration variables |
| OID | OID | PG_AUTHID.oid | ID of the role |
| roluseft | boolean | PG_AUTHID.roluseft | Whether the role can perform operations on foreign tables |
| nodegroup | Name | N/A | Name of the logical cluster associated with the role. If no logical cluster is associated, this column is left empty. |

18.3.129 PG_RULES

PG_RULES displays information about rewrite rules.

Table 18-200 PG_RULES columns

| Column | Type | Description |
|------------|------|--|
| schemaname | Name | Name of the schema that contains the table |
| tablename | Name | Name of the table the rule is for |
| rulename | Name | Rule name |
| definition | Text | Rule definition (a reconstructed creation command) |

18.3.130 PG_RUNNING_XACTS

PG_RUNNING_XACTS displays information about running transactions on the current node.

Table 18-201 PG_RUNNING_XACTS columns

| Column | Type | Description |
|-------------|---------|--|
| handle | Integer | Handle corresponding to the transaction in GTM |
| gxid | Xid | Transaction ID |
| state | tinyint | Transaction status (3 : prepared or 0 : starting) |
| node | Text | Node name |
| xmin | Xid | Minimum transaction ID xmin on the node |
| vacuum | boolean | Whether the current transaction is lazy vacuum |
| timeline | Bigint | Number of database restarts |
| prepare_xid | Xid | Transaction ID in the prepared status. If the status is not prepared , the value is 0 . |
| pid | Bigint | Thread ID corresponding to the transaction |
| next_xid | Xid | Transaction ID sent from a CN to a DN |

18.3.131 PG_SECLABELS

PG_SECLABELS displays information about security labels.

Table 18-202 PG_SECLABELS columns

| Column | Type | Reference | Description |
|--------------|---------|---------------------------------------|--|
| objoid | OID | Any OID column | OID of the object this security label pertains to |
| classoid | OID | PG_CLASS .oid | OID of the system table that contains the object |
| objsubid | Integer | N/A | For a security label on a table column, this is the column number (the objoid and classoid refer to the table itself). For all other object types, this column is 0 . |
| objtype | Text | N/A | Type of the object to which this label applies |
| objnamespace | OID | PG_NAMESPACE .oid | OID of the namespace for this object, if applicable; otherwise NULL. |
| objname | Text | N/A | Name of the object to which the label applies |
| provider | Text | PG_SECLABEL .provider | Label provider associated with this label |
| label | Text | PG_SECLABEL .label | Security label applied to this object |

18.3.132 PG_SEQUENCES

PG_SEQUENCES displays the sequence attributes on which the current user has permissions. This view is supported only by clusters of version 9.1.0 or later.

Table 18-203 PG_SEQUENCES columns

| Column | Type | Description |
|---------------|--------|--|
| schemaname | Name | Name of the namespace. |
| sequencename | Name | Name of the sequence |
| sequenceowner | Name | Owner of the sequence |
| start_value | Bigint | Start value of the sequence. |
| min_value | Bigint | Minimum value generated by the sequence. |
| max_value | Bigint | Maximum value generated by the sequence. |

| Column | Type | Description |
|--------------|---------|---|
| increment_by | Bigint | Amount by which the generated value increases each time in a sequence. |
| cycle | Boolean | If set to true , the sequence value restarts from the minimum value after reaching the maximum value. If set to false , the sequence value stops generating after reaching the maximum value. |
| cache_size | Bigint | Size of the sequence cache value. |
| last_value | Bigint | Most recently generated value of the sequence. |

18.3.133 PG_SESSION_WLMSTAT

PG_SESSION_WLMSTAT displays the corresponding load management information about the task currently executed by the user.

Table 18-204 PG_SESSION_WLMSTAT columns

| Column | Type | Description |
|------------|---------|--|
| datid | OID | OID of the database this backend is connected to |
| datname | Name | Name of the database the backend is connected to |
| threadid | Bigint | ID of the backend thread |
| processid | Integer | Thread PID of the backend |
| usesysid | OID | OID of the user who logged in to the backend |
| appname | Text | Name of the application that is connected to the backend |
| username | Name | Name of the user logged in to the backend |
| priority | Bigint | Priority of Cgroup where the statement is located |
| attribute | Text | Statement attributes <ul style="list-style-type: none">● Ordinary: default attribute of a statement before it is parsed by the database● Simple: simple statements● Complicated: complicated statements● Internal: internal statement of the database |
| block_time | Bigint | Pending duration of the statements by now (unit: s) |

| Column | Type | Description |
|------------------|---------|---|
| elapsed_time | Bigint | Actual execution duration of the statements by now (unit: s) |
| total_cpu_time | Bigint | Total CPU usage duration of the statement on the DN in the last period (unit: s) |
| cpu_skew_percent | Integer | CPU usage inclination ratio of the statement on the DN in the last period |
| statement_mem | Integer | Estimated memory required for statement execution. This column is reserved. |
| active_points | Integer | Number of concurrently active points occupied by the statement in the resource pool |
| dop_value | Integer | DOP value obtained by the statement from the resource pool |
| control_group | Text | Cgroup currently used by the statement |
| status | Text | Status of a statement, including: <ul style="list-style-type: none">• pending• running• finished (If enqueue is set to StoredProc or Transaction, this state indicates that only some of the jobs in the statement have been executed. This state persists until the finish of this statement.)• aborted: terminated unexpectedly• active: normal status except for those above• unknown: unknown status |
| enqueue | Text | Current queuing status of the statements, including: <ul style="list-style-type: none">• Global: global queuing.• Respool: resource pool queuing.• CentralQueue: queuing on the CCN• Transaction: being in a transaction block• StoredProc: being in a stored procedure• None: not in a queue• Forced None: being forcibly executed (transaction block statement or stored procedure statement are) because the statement waiting time exceeds the specified value |
| resource_pool | Name | Current resource pool where the statements are located. |

| Column | Type | Description |
|------------|------|--|
| query | Text | Text of this backend's most recent query. If state is active , this column shows the executing query. In all other states, it shows the last query that was executed. |
| isplana | Bool | In logical cluster mode, indicates whether a statement occupies the resources of other logical clusters. The default value is f , indicating that resources of other logical clusters are not occupied. |
| node_group | Text | Logical cluster of the user running the statement |
| lane | Text | Fast or slow lane for statement queries. <ul style="list-style-type: none">• fast: fast lane• slow: slow lane• none: not controlled |

18.3.134 PG_SESSION_IOSTAT

PG_SESSION_IOSTAT has been discarded in version 8.1.2 and is reserved for compatibility with earlier versions. This view is invalid in the current version. You can use [PGXC_WLM_SESSION_STATISTICS](#) to view load management information about jobs being executed on all CNs.

Table 18-205 PG_SESSION_IOSTAT columns

| Column | Type | Description |
|-------------|---------|---|
| query_id | Bigint | Job ID |
| mincurriops | Integer | Minimum I/O of the current job across DN |
| maxcurriops | Integer | Maximum I/O of the current job across DN |
| minpeakops | Integer | Minimum peak I/O of the current job across DN |
| maxpeakops | Integer | Maximum peak I/O of the current job across DN |
| io_limits | Integer | io_limits set for the job |
| io_priority | Text | io_priority set for the job |
| query | Text | Job |
| node_group | Text | Logical cluster of the user running the job |

18.3.135 PG_SETTINGS

PG_SETTINGS displays information about parameters of the running database.

Table 18-206 PG_SETTINGS columns

| Column | Type | Description |
|------------|---------|---|
| Name | Text | Parameter name |
| setting | Text | Current value of the parameter |
| unit | Text | Implicit unit of the parameter |
| category | Text | Logical group of the parameter |
| short_desc | Text | Brief description of the parameter |
| extra_desc | Text | Detailed description of the parameter |
| context | Text | Context of parameter values including internal, postmaster, sighup, backend, superuser, and user |
| vartype | Text | Parameter type. It can be bool , enum , integer , real , or string . |
| source | Text | Method of assigning the parameter value |
| min_val | Text | Minimum value of the parameter. If the parameter type is not numeric data, the value of this column is null. |
| max_val | Text | Maximum value of the parameter. If the parameter type is not numeric data, the value of this column is null. |
| enumvals | Text[] | Valid values of an enum-typed parameter. If the parameter type is not enum, the value of this column is null. |
| boot_val | Text | Default parameter value used upon the database startup |
| reset_val | Text | Default parameter value used upon the database reset |
| sourcefile | Text | Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is null. |
| sourceline | Integer | Row number of the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is null. |

18.3.136 PG_SHADOW

PG_SHADOW displays properties of all roles that are marked as **rolcanlogin** in **PG_AUTHID**.

This view is not readable to all users because it contains passwords. **PG_USER** is a publicly readable view on **PG_SHADOW** that blanks out the password column.

Table 18-207 PG_SHADOW columns

| Column | Type | Reference | Description |
|-------------|--------------------------|------------------------------------|--|
| username | Name | PG_AUTHID .rolname | User name |
| usesysid | OID | PG_AUTHID .oid | ID of a user |
| usecreatedb | boolean | - | Indicates that the user can create databases. |
| usesuper | boolean | - | Indicates that the user is an administrator. |
| usecatupd | boolean | - | Indicates that the user can update system catalogs. Even the system administrator cannot do this unless this column is true . |
| userepl | boolean | - | User can initiate streaming replication and put the system in and out of backup mode. |
| passwd | Text | - | Password (possibly encrypted); null if none. See PG_AUTHID for details about how encrypted passwords are stored. |
| valbegin | Timestamp with time zone | - | Account validity start time; null if no start time |
| valuntil | Timestamp with time zone | - | Password expiry time; null if no expiration |
| respool | Name | - | Resource pool used by the user |
| parent | OID | - | Parent resource pool |
| spacelimit | Text | - | The storage space of the permanent table. |

| Column | Type | Reference | Description |
|-----------------|---------|-----------|--|
| tempspacelimit | Text | - | The storage space of the temporary table. |
| spillspacelimit | Text | - | The operator disk flushing space. |
| useconfig | text[] | - | Session defaults for runtime configuration variables |

18.3.137 PG_SHARED_MEMORY_DETAIL

PG_SHARED_MEMORY_DETAIL displays usage information about all the shared memory contexts.

Table 18-208 PG_SHARED_MEMORY_DETAIL columns

| Column | Type | Description |
|-------------|----------|--|
| contextname | Text | Name of the memory context. |
| level | Smallint | Hierarchy of the memory context. |
| parent | Text | Parent memory context. |
| totalsize | Bigint | Total size of the shared memory, in bytes. |
| freesize | Bigint | Remaining size of the shared memory, in bytes. |
| usedsize | Bigint | Used size of the shared memory, in bytes. |

18.3.138 PG_STATS

PG_STATS displays the single-column statistics stored in the **pg_statistic** table.

Table 18-209 PG_STATS columns

| Column | Type | Reference | Description |
|------------|------|---------------------------------------|--|
| schemaname | Name | PG_NAMESPACE .nspname | Name of the schema that contains the table |
| tablename | Name | PG_CLASS .relname | Name of the table |
| attname | Name | PG_ATTRIBUTE .attname | Column name |

| Column | Type | Reference | Description |
|----------------------|----------|-----------|--|
| inherited | boolean | - | Includes inherited sub-columns if the value is true ; otherwise, indicates the column in a specified table. |
| null_frac | Real | - | Percentage of column entries that are null |
| avg_width | Integer | - | Average width in bytes of column's entries |
| n_distinct | Real | - | <ul style="list-style-type: none">Estimated number of distinct values in the column if the value is greater than 0Negative of the number of distinct values divided by the number of rows if the value is less than 0 <p>The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows.</p> <p>The positive form is used when the column seems to have a fixed number of possible values. For example, -1 indicates a unique column in which the number of distinct values is the same as the number of rows.</p> |
| n_dndistinct | Real | - | <p>Number of unique non-null data values in the dn1 column</p> <ul style="list-style-type: none">Exact number of distinct values if the value is greater than 0Negative of the number of distinct values divided by the number of rows if the value is less than 0 (For example, if the value of a column appears twice in average, set n_dndistinct=-0.5.)The number of distinct values is unknown if the value is 0. |
| most_commo
n_vals | anyarray | - | List of the most common values in a column. If this combination does not have the most common values, it will be NULL . |

| Column | Type | Reference | Description |
|------------------------|----------|-----------|---|
| most_common_freqs | real[] | - | List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals is NULL) |
| histogram_bounds | anyarray | - | List of values that divide the column's values into groups of equal proportion. The values in most_common_vals , if present, are omitted from this histogram calculation. This field is null if the field data type does not have a < operator or if the most_common_vals list accounts for the entire population. |
| correlation | Real | - | Statistical correlation between physical row ordering and logical ordering of the column values. It ranges from -1 to +1. When the value is near to -1 or +1, an index scan on the column is estimated to be cheaper than when it is near to zero, due to reduction of random access to the disk. This column is null if the column data type does not have a < operator. |
| most_common_elems | anyarray | - | Specifies a list of non-null element values most often appearing. |
| most_common_elem_freqs | real[] | - | Specifies a list of the frequencies of the most common element values. |
| elem_count_histogram | real[] | - | Histogram of the counts of distinct non-null element values. |
| partname | Name | - | Partition name. This column is available only in clusters of version 8.2.1.200 or later. |

18.3.139 PG_STAT_ACTIVITY

PG_STAT_ACTIVITY displays information about the current user's queries. If you have the rights of an administrator or the preset role, you can view all information about user queries.

Table 18-210 PG_STAT_ACTIVITY columns

| Column | Type | Description |
|------------------|--------------------------|---|
| datid | OID | OID of the database that the user session connects to in the backend |
| datname | Name | Name of the database that the user session connects to in the backend |
| pid | Bigint | Backend thread ID |
| lwtid | Integer | Lightweight thread ID |
| usesysid | OID | OID of the user logging in to the backend |
| username | Name | Username for logging in to the backend. |
| application_name | Text | Name of the application connected to the backend |
| client_addr | inet | IP address of the client connected to the backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used |
| backend_start | Timestamp with time zone | Startup time of the backend process, that is, the time when the client connects to the server. |
| xact_start | Timestamp with time zone | Time when the current transaction was started, or NULL if no transaction is active. If the current query is the first of its transaction, this column is equal to the query_start column. |
| query_start | Timestamp with time zone | Time when the currently active query was started, or if state is not active , when the last query was started |

| Column | Type | Description |
|--------------|--------------------------|---|
| state_change | Timestamp with time zone | Time for the last status change |
| waiting | boolean | The value is t if the backend is waiting for a lock or node. Otherwise, the value is f . |
| enqueue | Text | <p>Queuing status of a statement. Its value can be:</p> <ul style="list-style-type: none">• waiting in global queue: The statement is queuing in the global concurrent queue. The number of concurrent statements exceeds the value of max_active_statements configured for a single CN.• waiting in respool queue: The statement is queuing in the resource pool and the concurrency of simple jobs is limited. The main reason is that the concurrency of simple jobs exceeds the upper limit max_dop of the fast track.• waiting in ccn queue: The job is in the CCN queue, which may be global memory queuing, slow lane memory queuing, or concurrent queuing. The scenarios are:<ol style="list-style-type: none">1. The available global memory exceeds the upper limit, the job is queuing in the global memory queue.2. Concurrent requests on the slow lane in the resource pool exceed the upper limit, which is specified by active_statements.3. The slow lane memory of the resource pool exceeds the upper limit, that is, the estimated memory of concurrent jobs in the resource pool exceeds the upper limit specified by mem_percent.• Empty or no waiting queue: The statement is running. |

| Column | Type | Description |
|-----------------|--------|--|
| state | Text | <p>Current overall state of this backend. Its value can be:</p> <ul style="list-style-type: none">• active: The backend is executing queries.• idle: The backend is waiting for new client commands.• idle in transaction: The backend is in a transaction, but there is no statement being executed in the transaction.• idle in transaction (aborted): The backend is in a transaction, but there are statements failed in the transaction.• fastpath function call: The backend is executing a fast-path function.• disabled: This state is reported if track_activities is disabled in this backend. <p>NOTE
Common users can view only their own session status. The state information of other accounts is empty.</p> |
| resource_pool | Name | Resource pool used by the user |
| stmt_type | Text | Statement type |
| query_id | Bigint | ID of a query |
| query | Text | Text of the most recent query in this backend. If state is active , this column shows the running query. In all other states, it shows the last query that was executed. |
| connection_info | Text | A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database (for details, see connection_info). |

18.3.140 PG_STAT_ALL_INDEXES

PG_STAT_ALL_INDEXES displays statistics about all accesses to a specific index in the current database.

Indexes can be used via either simple index scans or "bitmap" index scans. Bitmap scans can combine the output of multiple indexes using AND or OR rules, but

combining independent row fetching with specific indexes is challenging. Consequently, a bitmap scan increases the index count in **pg_stat_all_indexes.idx_tup_read** and the table count in **pg_stat_all_tables.idx_tup_fetch**, while having no effect on **pg_stat_all_indexes.idx_tup_fetch**.

Table 18-211 PG_STAT_ALL_INDEXES columns

| Column | Type | Description |
|---------------|--------|---|
| relid | OID | OID of the table for this index. |
| indexrelid | OID | OID of this index. |
| schemaname | Name | Name of the schema this index is in. |
| relname | Name | Name of the table for this index. |
| indexrelname | Name | Name of this index. |
| idx_scan | Bigint | Number of index scans initiated on this index. |
| idx_tup_read | Bigint | Number of index entries returned by scans on this index. |
| idx_tup_fetch | Bigint | Number of live table rows fetched by simple index scans using this index. |

18.3.141 PG_STAT_ALL_TABLES

PG_STAT_ALL_TABLES displays statistics about accesses to tables in the current database, including TOAST tables.

Table 18-212 PG_STAT_ALL_TABLES columns

| Column | Type | Description |
|---------------|--------|--|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of the table |
| seq_scan | Bigint | Number of sequential scans started on the table |
| seq_tup_read | Bigint | Number of rows that have live data fetched by sequential scans |
| idx_scan | Bigint | Number of index scans |
| idx_tup_fetch | Bigint | Number of rows that have live data fetched by index scans |
| n_tup_ins | Bigint | Number of rows inserted |

| Column | Type | Description |
|-------------------|--------------------------|--|
| n_tup_upd | Bigint | Number of rows updated |
| n_tup_del | Bigint | Number of rows deleted |
| n_tup_hot_upd | Bigint | Number of rows updated by HOT (no separate index update is required) |
| n_live_tup | Bigint | Estimated number of live rows |
| n_dead_tup | Bigint | Estimated number of dead rows |
| last_vacuum | Timestamp with time zone | Last time at which this table was manually vacuumed (excluding VACUUM FULL) |
| last_autovacuum | Timestamp with time zone | Last time at which this table was automatically vacuumed |
| last_analyze | Timestamp with time zone | Time when the last non-automatic rolling analysis is performed. |
| last_autoanalyze | Timestamp with time zone | Time when the last automatic rolling analysis is performed. |
| vacuum_count | Bigint | Number of vacuum operations (excluding VACUUM FULL) |
| autovacuum_count | Bigint | Number of autovacuum operations |
| analyze_count | Bigint | Number of ANALYZE operations |
| autoanalyze_count | Bigint | Number of autoanalyze operations |
| last_data_changed | Timestamp with time zone | Last time at which this table was updated (by INSERT/UPDATE/DELETE or EXCHANGE/TRUNCATE/DROP <i>partition</i>). This column is recorded only on the local CN. |

Example

Query the last data change time in the **table_test** table:

```
SELECT last_data_changed FROM PG_STAT_ALL_TABLES WHERE relname ='table_test';
      last_data_changed
-----
2024-03-27 10:28:16.277136+08
(1 row)
```


18.3.142 PG_STAT_BAD_BLOCK

PG_STAT_BAD_BLOCK displays statistics about page or CU verification failures after a node is started.

Table 18-213 PG_STAT_BAD_BLOCK columns

| Column | Type | Description |
|--------------|--------------------------|----------------------------------|
| nodename | Text | Node name. |
| databaseid | Integer | Database OID. |
| tablespaceid | Integer | Tablespace OID. |
| relfilenode | Integer | File object ID. |
| forknum | Integer | File type. |
| error_count | Integer | Number of verification failures. |
| first_time | Timestamp with time zone | Time of the first occurrence. |
| last_time | Timestamp with time zone | Time of the latest occurrence. |

18.3.143 PG_STAT_BGWRITER

PG_STAT_BGWRITER displays statistics about the background writer process's activity.

Table 18-214 PG_STAT_BGWRITER columns

| Column | Type | Description |
|-----------------------|------------------|---|
| checkpoints_timed | Bigint | Number of scheduled checkpoints that have been performed. |
| checkpoints_req | Bigint | Number of requested checkpoints that have been performed. |
| checkpoint_write_time | Double precision | Time spent on writing files to the disk during checkpoints, in milliseconds. |
| checkpoint_sync_time | Double precision | Time spent on synchronizing data to the disk during checkpoints, in milliseconds. |
| buffers_checkpoint | Bigint | Number of buffers written during checkpoints. |

| Column | Type | Description |
|-----------------------|--------------------------|--|
| buffers_clean | Bigint | Number of buffers written by the background writer. |
| maxwritten_clean | Bigint | Number of times the background writer stopped a cleaning scan because too many buffers were written. |
| buffers_backend | Bigint | Number of buffers written directly by a backend |
| buffers_backend_fsync | Bigint | Number of times that the backend has to execute fsync . |
| buffers_alloc | Bigint | Number of buffers allocated. |
| stats_reset | Timestamp with time zone | Time at which these statistics were reset. |

18.3.144 PG_STAT_DATABASE

PG_STAT_DATABASE displays the status and statistics of each database on the current node.

Table 18-215 PG_STAT_DATABASE columns

| Column | Type | Description |
|---------------|---------|---|
| datid | OID | Database OID |
| datname | Name | Database name |
| numbackends | Integer | Number of backends currently connected to this database on the current node. This is the only column in this view that reflects the current state value. All columns return the accumulated value since the last reset. |
| xact_commit | Bigint | Number of transactions in this database that have been committed on the current node |
| xact_rollback | Bigint | Number of transactions in this database that have been rolled back on the current node |
| blks_read | Bigint | Number of disk blocks read in this database on the current node |
| blks_hit | Bigint | Number of disk blocks found in the buffer cache on the current node, that is, the number of blocks hit in the cache. (This only includes hits in the DWS buffer cache, not in the file system cache.) |

| Column | Type | Description |
|----------------|--------------------------|---|
| tup_returned | Bigint | Number of rows returned by queries in this database on the current node |
| tup_fetched | Bigint | Number of rows fetched by queries in this database on the current node |
| tup_inserted | Bigint | Number of rows inserted in this database on the current node |
| tup_updated | Bigint | Number of rows updated in this database on the current node |
| tup_deleted | Bigint | Number of rows deleted from this database on the current node |
| conflicts | Bigint | Number of queries canceled due to database recovery conflicts on the current node (conflicts occurring only on the standby server). For details, see PG_STAT_DATABASE_CONFLICTS . |
| temp_files | Bigint | Number of temporary files created by this database on the current node. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the log_temp_files setting. |
| temp_bytes | Bigint | Size of temporary files written to this database on the current node. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting. |
| deadlocks | Bigint | Number of deadlocks in this database on the current node |
| blk_read_time | Double precision | Time spent reading data file blocks by backends in this database on the current node, in milliseconds |
| blk_write_time | Double precision | Time spent writing into data file blocks by backends in this database on the current node, in milliseconds |
| stats_reset | Timestamp with time zone | Time when the database statistics are reset on the current node |

18.3.145 PG_STAT_DATABASE_CONFLICTS

PG_STAT_DATABASE_CONFLICTS displays statistics about database conflicts.

Table 18-216 PG_STAT_DATABASE_CONFLICTS columns

| Column | Type | Description |
|------------------|--------|------------------------------------|
| datid | OID | Database OID. |
| datname | Name | Database name. |
| confl_tablespace | Bigint | Number of conflicting tablespaces. |
| confl_lock | Bigint | Number of conflicting locks. |
| confl_snapshot | Bigint | Number of conflicting snapshots. |
| confl_bufferpin | Bigint | Number of conflicting buffers. |
| confl_deadlock | Bigint | Number of conflicting deadlocks. |

18.3.146 PG_STATS_EXT_EXPRS

PG_STATS_EXT_EXPRS provides statistics about expressions stored in the **PG_STATISTIC_DATA** table. This view is supported only by clusters of version 8.2.1.200 or later.

Table 18-217 PG_STATS_EXT_EXPRS columns

| Column | Type | Reference | Description |
|---------------------------|---------|--|---|
| schemaname | Name | PG_NAMESP
ACE .nspname | Name of the schema that contains the table |
| tablename | Name | PG_CLASS .rel
name | Table name |
| statistics_sche
maname | Name | - | Name of the schema where expression objects whose statistics are collected is located |
| statistics_name | Name | - | Name of the expression object whose statistics are collected |
| statistics_owner | Name | - | Owner of the expression object whose statistics are collected |
| expr | Text | - | Expression |
| inherited | boolean | - | If the value is true , the inherited subcolumns are included. If the value is false , only the columns in a specified table are included. |

| Column | Type | Reference | Description |
|-----------------------|----------|-----------|---|
| null_frac | Real | - | Percentage of column entries that are null |
| avg_width | Integer | - | Average width in bytes of column's entries |
| n_distinct | Real | - | <ul style="list-style-type: none">• If the value is greater than 0, it indicates the estimated number of distinct values in the column.• If the value is less than 0, it indicates that the number of distinct values divided by the number of rows. <p>The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows.</p> <p>The positive form is used when the column seems to have a fixed number of possible values. For example, -1 indicates a unique column in which the number of distinct values is the same as the number of rows.</p> |
| n_dndistinct | Real | - | <p>Number of unique non-null data values in the dn1 column</p> <ul style="list-style-type: none">• Exact number of distinct values if the value is greater than 0• Negative of the number of distinct values divided by the number of rows if the value is less than 0 (For example, if the value of a column appears twice in average, set n_dndistinct=-0.5.)• The number of distinct values is unknown if the value is 0. |
| most_commo
n_vals | anyarray | - | List of the most common values in a column. If this combination does not have the most common values, it will be NULL . |
| most_commo
n_freqs | real[] | - | List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals is NULL) |

| Column | Type | Reference | Description |
|------------------------|----------|-----------|---|
| histogram_bounds | anyarray | - | List of values that divide the column's values into groups of equal proportion. The values in most_common_vals , if present, are omitted from this histogram calculation. This field is null if the field data type does not have a < operator or if the most_common_vals list accounts for the entire population. |
| correlation | Real | - | Statistical correlation between physical row ordering and logical ordering of the column values. It ranges from -1 to +1. When the value is near to -1 or +1, an index scan on the column is estimated to be cheaper than when it is near to zero, due to reduction of random access to the disk. This column is null if the column data type does not have a < operator. |
| most_common_elems | anyarray | - | Specifies a list of non-null element values most often appearing. |
| most_common_elem_freqs | real[] | - | Specifies a list of the frequencies of the most common element values. |
| elem_count_histogram | real[] | - | Specifies a histogram of the counts of distinct non-null element values. |

18.3.147 PG_STAT_GET_MEM_MBYTES_RESERVED

PG_STAT_GET_MEM_MBYTES_RESERVED displays the current activity information of a thread stored in memory. You need to specify the thread ID (pid in [PG_STAT_ACTIVITY](#)) for query. If the thread ID is set to 0, the current thread ID is used. For example:

```
SELECT pg_stat_get_mem_mbytes_reserved(0);
```

Table 18-218 PG_STAT_GET_MEM_MBYTES_RESERVED columns

| Column | Description |
|---------------|-------------------------------------|
| ConnectInfo | Connection information. |
| ParctlManager | Concurrency management information. |

| Column | Description |
|----------------------|-----------------------------------|
| GeneralParams | Basic parameter information. |
| GeneralParams RPDATA | Basic resource pool information. |
| ExceptionManager | Exception management information. |
| CollectInfo | Collection information. |
| GeneralInfo | Basic information. |
| ParctlState | Concurrency status information. |
| CPU INFO | CPU information. |
| ControlGroup | Cgroup information. |
| IOWSTATE | I/O status information. |

18.3.148 PG_STAT_USER_FUNCTIONS

PG_STAT_USER_FUNCTIONS displays user-defined function status information in the namespace. (The language of the function is non-internal language.)

Table 18-219 PG_STAT_USER_FUNCTIONS columns

| Column | Type | Description |
|------------|------------------|--|
| funcid | OID | Function OID |
| schemaname | Name | Schema name |
| funcname | Name | Name of the function |
| calls | Bigint | Number of times this function has been called |
| total_time | Double precision | Total time spent in this function and all other functions called by it |
| self_time | Double precision | Total time spent in this function itself, excluding other functions called by it |

18.3.149 PG_STAT_USER_INDEXES

PG_STAT_USER_INDEXES displays information about the index status of user-defined ordinary tables and TOAST tables.

Table 18-220 PG_STAT_USER_INDEXES columns

| Column | Type | Description |
|---------------|--------|---|
| relid | OID | Table OID for the index |
| indexrelid | OID | OID of this index |
| schemaname | Name | Name of the schema this index is in |
| relname | Name | Name of the table for this index |
| indexrelname | Name | Name of this index |
| idx_scan | Bigint | Number of index scans |
| idx_tup_read | Bigint | Number of index entries returned by scans on this index |
| idx_tup_fetch | Bigint | Number of rows that have live data fetched by index scans |

18.3.150 PG_STAT_USER_TABLES

PG_STAT_USER_TABLES displays status information about user-defined ordinary tables and TOAST tables in all namespaces.

Table 18-221 PG_STAT_USER_TABLES columns

| Column | Type | Description |
|---------------|--------|---|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of a table |
| seq_scan | Bigint | Number of sequential scans started on the table |
| seq_tup_read | Bigint | Number of rows that have live data fetched by sequential scans |
| idx_scan | Bigint | Number of index scans |
| idx_tup_fetch | Bigint | Number of rows that have live data fetched by index scans |
| n_tup_ins | Bigint | Number of rows inserted |
| n_tup_upd | Bigint | Number of rows updated |
| n_tup_del | Bigint | Number of rows deleted |
| n_tup_hot_upd | Bigint | Number of rows updated by HOT (no separate index update is required) |

| Column | Type | Description |
|-------------------|--------------------------|---|
| n_live_tup | Bigint | Estimated number of live rows |
| n_dead_tup | Bigint | Estimated number of dead rows |
| last_vacuum | Timestamp with time zone | Last time at which this table was manually vacuumed (excluding VACUUM FULL) |
| last_autovacuum | Timestamp with time zone | Time of the last AUTOVACUUM |
| last_analyze | Timestamp with time zone | Time when the last non-automatic rolling analysis is performed. |
| last_autoanalyze | Timestamp with time zone | Time when the last automatic rolling analysis is performed. |
| vacuum_count | Bigint | Number of vacuum operations (excluding VACUUM FULL) |
| autovacuum_count | Bigint | Number of autovacuum operations |
| analyze_count | Bigint | Number of analyze operations |
| autoanalyze_count | Bigint | Number of autoanalyze operations |

18.3.151 PG_STAT_REPLICATION

PG_STAT_REPLICATION displays information about log synchronization status, such as the locations of the sender sending logs and the receiver receiving logs.

Table 18-222 PG_STAT_REPLICATION columns

| Column | Type | Description |
|------------------|--------|--------------------|
| pid | Bigint | PID of the thread. |
| usesysid | OID | User system ID. |
| username | Name | Username. |
| application_name | Text | Program name. |
| client_addr | inet | Client address. |
| client_hostname | Text | Client name. |

| Column | Type | Description |
|--------------------------|--------------------------|--|
| client_port | Integer | Client port number. |
| backend_start | Timestamp with time zone | Program start time. |
| state | Text | Log replication state (catch-up or consistent streaming). |
| sender_sent_location | Text | Location where the sender sends logs. |
| receiver_write_location | Text | Location where the receiver writes logs. |
| receiver_flush_location | Text | Location where the receiver flushes logs. |
| receiver_replay_location | Text | Location where the receiver replays logs. |
| sync_priority | Integer | Priority of synchronous duplication (0 indicates asynchronization). |
| sync_state | Text | Synchronization state (asynchronous duplication, synchronous duplication, or potential synchronization). |

18.3.152 PG_STAT_SYS_INDEXES

PG_STAT_SYS_INDEXES displays the index status information about all the system catalogs in the **pg_catalog** and **information_schema** schemas.

Table 18-223 PG_STAT_SYS_INDEXES columns

| Column | Type | Description |
|--------------|--------|--|
| relid | OID | Table OID for the index. |
| indexrelid | OID | OID of this index. |
| schemaname | Name | Name of the schema this index is in. |
| relname | Name | Name of the table for this index. |
| indexrelname | Name | Name of this index. |
| idx_scan | Bigint | Number of index scans. |
| idx_tup_read | Bigint | Number of index entries returned by scans on this index. |

| Column | Type | Description |
|---------------|--------|--|
| idx_tup_fetch | Bigint | Number of rows that have live data fetched by index scans. |

18.3.153 PG_STAT_SYS_TABLES

PG_STAT_SYS_TABLES displays the statistics about the system catalogs of all the namespaces in **pg_catalog** and **information_schema** schemas.

Table 18-224 PG_STAT_SYS_TABLES columns

| Column | Type | Description |
|-----------------|--------------------------|---|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of a table |
| seq_scan | Bigint | Number of sequential scans started on the table |
| seq_tup_read | Bigint | Number of rows that have live data fetched by sequential scans |
| idx_scan | Bigint | Number of index scans |
| idx_tup_fetch | Bigint | Number of rows that have live data fetched by index scans |
| n_tup_ins | Bigint | Number of rows inserted |
| n_tup_upd | Bigint | Number of rows updated |
| n_tup_del | Bigint | Number of rows deleted |
| n_tup_hot_upd | Bigint | Number of rows updated by HOT (no separate index update is required) |
| n_live_tup | Bigint | Estimated number of live rows |
| n_dead_tup | Bigint | Estimated number of dead rows |
| last_vacuum | Timestamp with time zone | Last time at which this table was manually vacuumed (excluding VACUUM FULL) |
| last_autovacuum | Timestamp with time zone | Last time at which this table was automatically vacuumed |
| last_analyze | Timestamp with time zone | Last time at which this table was analyzed |

| Column | Type | Description |
|-------------------|--------------------------|---|
| last_autoanalyze | Timestamp with time zone | Last time at which this table was automatically analyzed |
| vacuum_count | Bigint | Number of vacuum operations (excluding VACUUM FULL) |
| autovacuum_count | Bigint | Number of autovacuum operations |
| analyze_count | Bigint | Number of analyze operations |
| autoanalyze_count | Bigint | Number of autoanalyze operations |

18.3.154 PG_STAT_XACT_ALL_TABLES

PG_STAT_XACT_ALL_TABLES displays the transaction status information about all ordinary tables and TOAST tables in the namespaces.

Table 18-225 PG_STAT_XACT_ALL_TABLES columns

| Column | Type | Description |
|---------------|--------|---|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of a table |
| seq_scan | Bigint | Number of sequential scans started on the table |
| seq_tup_read | Bigint | Number of live rows fetched by sequential scans |
| idx_scan | Bigint | Number of index scans started on the table |
| idx_tup_fetch | Bigint | Number of live rows fetched by index scans |
| n_tup_ins | Bigint | Number of rows inserted |
| n_tup_upd | Bigint | Number of rows updated |
| n_tup_del | Bigint | Number of rows deleted |
| n_tup_hot_upd | Bigint | Number of rows with HOT updates (no separate index update is required). |

18.3.155 PG_STAT_XACT_SYS_TABLES

PG_STAT_XACT_SYS_TABLES displays the transaction status information of the system catalog in the namespace.

Table 18-226 PG_STAT_XACT_SYS_TABLES columns

| Column | Type | Description |
|--------------------|--------|---|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Table name |
| seq_scan | Bigint | Number of sequential scans started on the table |
| seq_tup_read | Bigint | Number of live rows fetched by sequential scans |
| idx_scan | Bigint | Number of index scans started on the table |
| idx_tup_fetch | Bigint | Number of live rows fetched by index scans |
| n_tup_ins | Bigint | Number of rows inserted |
| n_tup_upd | Bigint | Number of rows updated |
| n_tup_del | Bigint | Number of rows deleted |
| n_tup_hot_upd
d | Bigint | Number of rows with HOT updates (no separate index update is required). |

18.3.156 PG_STAT_XACT_USER_FUNCTIONS

PG_STAT_XACT_USER_FUNCTIONS displays statistics about function execution.

Table 18-227 PG_STAT_XACT_USER_FUNCTIONS columns

| Column | Type | Description |
|------------|------------------|--|
| funcid | OID | Function OID |
| schemaname | Name | Schema name |
| funcname | Name | Name of the function |
| calls | Bigint | Number of times this function has been called |
| total_time | Double precision | Total time spent in this function and all other functions called by it |
| self_time | Double precision | Total time spent in this function itself, excluding other functions called by it |

18.3.157 PG_STAT_XACT_USER_TABLES

PG_STAT_XACT_USER_TABLES displays the transaction status information of the user table in the namespace.

Table 18-228 PG_STAT_XACT_USER_TABLES columns

| Column | Type | Description |
|--------------------|--------|---|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of a table |
| seq_scan | Bigint | Number of sequential scans started on the table |
| seq_tup_read | Bigint | Number of live rows fetched by sequential scans |
| idx_scan | Bigint | Number of index scans started on the table |
| idx_tup_fetch | Bigint | Number of live rows fetched by index scans |
| n_tup_ins | Bigint | Number of rows inserted |
| n_tup_upd | Bigint | Number of rows updated |
| n_tup_del | Bigint | Number of rows deleted |
| n_tup_hot_upd
d | Bigint | Number of rows with HOT updates (no separate index update is required). |

18.3.158 PG_STATIO_ALL_INDEXES

PG_STATIO_ALL_INDEXES displays I/O statistics of all indexes in the current database.

Table 18-229 PG_STATIO_ALL_INDEXES columns

| Column | Type | Description |
|---------------|--------|---|
| relid | OID | OID of the index table |
| indexrelid | OID | OID of this index |
| schemaname | Name | Name of the schema this index is in |
| relname | Name | Name of the table for this index |
| indexrelname | Name | Name of this index |
| idx_blks_read | Bigint | Number of disk blocks read from the index |
| idx_blks_hit | Bigint | Number of buffer hits in this index |

18.3.159 PG_STATIO_ALL_SEQUENCES

PG_STATIO_ALL_SEQUENCES displays the sequence information in the current database and the I/O statistics of a specified sequence.

Table 18-230 PG_STATIO_ALL_SEQUENCES columns

| Column | Type | Description |
|------------|--------|--|
| relid | OID | OID of this sequence |
| schemaname | Name | Name of the schema this sequence is in |
| relname | Name | Name of this sequence |
| blks_read | Bigint | Number of disk blocks read from the sequence |
| blks_hit | Bigint | Number of buffer hits in this sequence |

18.3.160 PG_STATIO_ALL_TABLES

PG_STATIO_ALL_TABLES displays I/O statistics about all tables (including TOAST tables) in the current database.

Table 18-231 PG_STATIO_ALL_TABLES columns

| Column | Type | Description |
|-----------------|--------|--|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of a table |
| heap_blks_read | Bigint | Number of disks read from this table |
| heap_blks_hit | Bigint | Number of buffer hits in this table |
| idx_blks_read | Bigint | Number of disk blocks read from the index in this table |
| idx_blks_hit | Bigint | Number of buffer hits in all indexes on this table |
| toast_blks_read | Bigint | Number of disk blocks read from the TOAST table (if any) in this table |
| toast_blks_hit | Bigint | Number of buffer hits in the TOAST table (if any) in this table |

| Column | Type | Description |
|----------------|--------|--|
| tidx_blks_read | Bigint | Number of disk blocks read from the TOAST table index (if any) in this table |
| tidx_blks_hit | Bigint | Number of buffer hits in the TOAST table index (if any) in this table |

18.3.161 PG_STATIO_SYS_INDEXES

PG_STATIO_SYS_INDEXES displays the I/O status information about all system catalog indexes in the namespace.

Table 18-232 PG_STATIO_SYS_INDEXES columns

| Column | Type | Description |
|---------------|--------|--|
| relid | OID | Table OID for the index. |
| indexrelid | OID | OID of this index. |
| schemaname | Name | Name of the schema of the index. |
| relname | Name | Name of the table for this index. |
| indexrelname | Name | Name of this index. |
| idx_blks_read | Bigint | Number of disk blocks read from the index. |
| idx_blks_hit | Bigint | Number of buffer hits in this index. |

18.3.162 PG_STATIO_SYS_SEQUENCES

PG_STATIO_SYS_SEQUENCES displays the I/O status information about all the system sequences in the namespace.

Table 18-233 PG_STATIO_SYS_SEQUENCES columns

| Column | Type | Description |
|------------|--------|--|
| relid | OID | OID of this sequence |
| schemaname | Name | Name of the schema this sequence is in |
| relname | Name | Name of this sequence |
| blks_read | Bigint | Number of disk blocks read from the sequence |
| blks_hit | Bigint | Number of buffer hits in this sequence |

18.3.163 PG_STATIO_SYS_TABLES

PG_STATIO_SYS_TABLES displays the I/O status information about all the system catalogs in the namespace.

Table 18-234 PG_STATIO_SYS_TABLES columns

| Column | Type | Description |
|-----------------|--------|--|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of a table |
| heap_blks_read | Bigint | Number of disk blocks read from this table |
| heap_blks_hit | Bigint | Number of buffer hits in this table |
| idx_blks_read | Bigint | Number of disk blocks read from the index in this table |
| idx_blks_hit | Bigint | Number of buffer hits in all indexes on this table |
| toast_blks_read | Bigint | Number of disk blocks read from the TOAST table (if any) in this table |
| toast_blks_hit | Bigint | Number of buffer hits in the TOAST table (if any) in this table |
| tidx_blks_read | Bigint | Number of disk blocks read from the TOAST table index (if any) in this table |
| tidx_blks_hit | Bigint | Number of buffer hits in the TOAST table index (if any) in this table |

18.3.164 PG_STATIO_USER_INDEXES

PG_STATIO_USER_INDEXES displays the I/O status information about all the user relationship table indexes in the namespace.

Table 18-235 PG_STATIO_USER_INDEXES columns

| Column | Type | Description |
|------------|------|-------------------------------------|
| relid | OID | OID of the table for this index |
| indexrelid | OID | OID of this index |
| schemaname | Name | Name of the schema this index is in |
| relname | Name | Name of the table for this index |

| Column | Type | Description |
|---------------|--------|---|
| indexrelname | Name | Name of this index |
| idx_blks_read | Bigint | Number of disk blocks read from the index |
| idx_blks_hit | Bigint | Number of buffer hits in this index |

18.3.165 PG_STATIO_USER_SEQUENCES

PG_STATIO_USER_SEQUENCES displays the I/O status information about all the user relation table sequences in the namespace.

Table 18-236 PG_STATIO_USER_SEQUENCES columns

| Column | Type | Description |
|------------|--------|--|
| relid | OID | OID of this sequence |
| schemaname | Name | Name of the schema this sequence is in |
| relname | Name | Name of this sequence |
| blks_read | Bigint | Number of disk blocks read from the sequence |
| blks_hit | Bigint | Cache hits in the sequence |

18.3.166 PG_STATIO_USER_TABLES

PG_STATIO_USER_TABLES displays the I/O status information about all the user relation tables in the namespace.

Table 18-237 PG_STATIO_USER_TABLES columns

| Column | Type | Description |
|----------------|--------|---|
| relid | OID | Table OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Name of a table |
| heap_blks_read | Bigint | Number of disk blocks read from this table |
| heap_blks_hit | Bigint | Number of buffer hits in this table |
| idx_blks_read | Bigint | Number of disk blocks read from the index in this table |

| Column | Type | Description |
|-----------------|--------|--|
| idx_blks_hit | Bigint | Number of buffer hits in all indexes on this table |
| toast_blks_read | Bigint | Number of disk blocks read from the TOAST table (if any) in this table |
| toast_blks_hit | Bigint | Number of buffer hits in the TOAST table (if any) in this table |
| tidx_blks_read | Bigint | Number of disk blocks read from the TOAST table index (if any) in this table |
| tidx_blks_hit | Bigint | Number of buffer hits in the TOAST table index (if any) in this table |

18.3.167 PG_THREAD_WAIT_STATUS

PG_THREAD_WAIT_STATUS allows you to test the block waiting status about the backend thread and auxiliary thread of the current instance.

Table 18-238 PG_THREAD_WAIT_STATUS columns

| Column | Type | Description |
|-------------|---------|---|
| node_name | Text | Current node name |
| db_name | Text | Database name |
| thread_name | Text | Thread name |
| query_id | Bigint | Query ID. It is equivalent to debug_query_id . |
| tid | Bigint | Thread ID of the current thread |
| lwtid | Integer | Lightweight thread ID of the current thread |
| ptid | Integer | Parent thread of the streaming thread |
| tlevel | Integer | Level of the streaming thread |
| smpid | Integer | Concurrent thread ID |
| wait_status | Text | Waiting status of the current thread. For details about the waiting status, see Table 18-239 . |
| wait_event | Text | If wait_status is acquire lock , acquire lwlock , or wait io , this column describes the lock, lightweight lock, and I/O information, respectively. If wait_status is not any of the three values, this column is empty. |

The waiting statuses in the **wait_status** column are as follows:

Table 18-239 Waiting status list

| Value | Description |
|---|---|
| none | Waiting for no event |
| acquire lock | Waiting for locking until the locking succeeds or times out |
| acquire lwlock | Waiting for a lightweight lock |
| wait io | Waiting for I/O completion |
| wait cmd | Waiting for network communication packet read to complete |
| wait pooler get conn | Waiting for pooler to obtain the connection |
| wait pooler abort conn | Waiting for pooler to terminate the connection |
| wait pooler clean conn | Waiting for pooler to clear connections |
| pooler create conn: [nodename], total N | Waiting for the pooler to set up a connection. The connection is being established with the node specified by <i>nodename</i> , and there are <i>N</i> connections waiting to be set up. |
| get conn | Obtaining the connection to other nodes |
| set cmd: [nodename] | Waiting for running the SET , RESET , TRANSACTION BLOCK LEVEL PARA SET , or SESSION LEVEL PARA SET statement on the connection. The statement is being executed on the node specified by <i>nodename</i> . |
| cancel query | Canceling the SQL statement that is being executed through the connection |
| stop query | Stopping the query that is being executed through the connection |
| wait node: [nodename](plevel), total N, [phase] | <p>Waiting for receiving the data from a connected node. The thread is waiting for the data from the plevel thread of the node specified by <i>nodename</i>. The data of <i>N</i> connections is waiting to be returned. If <i>phase</i> is included, the possible phases are as follows:</p> <ul style="list-style-type: none">• begin: The transaction is being started.• commit: The transaction is being committed.• rollback: The transaction is being rolled back. |

| Value | Description |
|---|--|
| wait transaction sync: xid | Waiting for synchronizing the transaction specified by <i>xid</i> |
| wait wal sync | Waiting for the completion of wal log of synchronization from the specified LSN to the standby instance |
| wait data sync | Waiting for the completion of data page synchronization to the standby instance |
| wait data sync queue | Waiting for putting the data pages that are in the row storage or the CU in the column storage into the synchronization queue |
| flush data: [nodename](plevel), [phase] | Waiting for sending data to the plevel thread of the node specified by <i>nodename</i> . If <i>phase</i> is included, the possible phase is wait quota , indicating that the current communication flow is waiting for the quota value. |
| stream get conn: [nodename], total N | Waiting for connecting to the consumer object of the node specified by <i>nodename</i> when the stream flow is initialized. There are <i>N</i> consumers waiting to be connected. |
| wait producer ready: [nodename] (plevel), total N | Waiting for each producer to be ready when the stream flow is initialized. The thread is waiting for the procedure of the plevel thread on the <i>nodename</i> node to be ready. There are <i>N</i> producers waiting to be ready. |
| synchronize quit | Waiting for the threads in the stream thread group to quit when the stream plan ends |
| nodegroup destroy | Waiting for destroying the stream node group when the stream plan ends |
| wait active statement | Waiting for job execution under resource and load control. |
| wait global queue | Waiting for job execution. The job is queuing in the global queue. |
| wait respool queue | Waiting for job execution. The job is queuing in the resource pool. |
| wait ccn queue | Waiting for job execution. The job is queuing on the central coordinator node (CCN). |
| gtm connect | Waiting for connecting to GTM. |
| gtm get gxid | Wait for obtaining xids from GTM. |

| Value | Description |
|--|---|
| gtm get snapshot | Wait for obtaining transaction snapshots from GTM. |
| gtm begin trans | Waiting for GTM to start a transaction. |
| gtm commit trans | Waiting for GTM to commit a transaction. |
| gtm rollback trans | Waiting for GTM to roll back a transaction. |
| gtm create sequence | Waiting for GTM to create a sequence. |
| gtm alter sequence | Waiting for GTM to modify a sequence. |
| gtm get sequence val | Waiting for obtaining the next value of a sequence from GTM. |
| gtm set sequence val | Waiting for GTM to set a sequence value. |
| gtm drop sequence | Waiting for GTM to delete a sequence. |
| gtm rename sequence | Waiting for GTM to rename a sequence. |
| analyze: [relname], [phase] | The thread is doing ANALYZE to the <i>relname</i> table. If <i>phase</i> is included, the possible phase is autovacuum , indicating that the database automatically enables the AutoVacuum thread to execute ANALYZE . |
| vacuum: [relname], [phase] | The thread is doing VACUUM to the <i>relname</i> table. If <i>phase</i> is included, the possible phase is autovacuum , indicating that the database automatically enables the AutoVacuum thread to execute VACUUM . |
| vacuum full: [relname] | The thread is doing VACUUM FULL to the <i>relname</i> table. |
| create index | An index is being created. |
| HashJoin - [build hash write file] | The HashJoin operator is being executed. In this phase, you need to pay attention to the execution time-consuming. <ul style="list-style-type: none">• build hash: The HashJoin operator is creating a hash table.• write file: The HashJoin operator is writing data to disks. |
| HashAgg - [build hash write file] | The HashAgg operator is being executed. In this phase, you need to pay attention to the execution time-consuming. <ul style="list-style-type: none">• build hash: The HashAgg operator is creating a hash table.• write file: The HashAgg operator is writing data to disks. |

| Value | Description |
|--|--|
| HashSetop - [build hash write file] | The HashSetop operator is being executed. In this phase, you need to pay attention to the execution time-consuming. <ul style="list-style-type: none">• build hash: The HashSetop operator is creating a hash table.• write file: The HashSetop operator is writing data to disks. |
| Sort Sort - write file | The Sort operator is being executed. write file indicates that the Sort operator is writing data to disks. |
| Material Material - write file | The Material operator is being executed. write file indicates that the Material operator is writing data to disks. |
| wait sync consumer next step | The consumer (receive end) synchronously waits for the next iteration. |
| wait sync producer next step | The producer (transmit end) synchronously waits for the next iteration. |
| wait agent release | The current agent is being released (supported by 8.1.2 and later versions). |
| wait stream task | The stream thread is waiting for being reused (supported by 8.1.2 and later versions). |

If **wait_status** is **acquire lwlock**, **acquire lock**, or **wait io**, there is an event performing I/O operations or waiting for obtaining the corresponding lightweight lock or transaction lock.

The following table describes the corresponding wait events when **wait_status** is **acquire lwlock**. (If **wait_event** is **extension**, the lightweight lock is dynamically allocated and is not monitored.)

Table 18-240 List of wait events corresponding to lightweight locks

| wait_event | Description |
|----------------|---|
| ShmemIndexLock | Used to protect the primary index table, a hash table, in shared memory |
| OidGenLock | Used to prevent different threads from generating the same OID |
| XidGenLock | Used to prevent two transactions from obtaining the same XID |

| wait_event | Description |
|-----------------------------|---|
| ProcArrayLock | Used to prevent concurrent access to or concurrent modification on the ProcArray shared array |
| SInvalReadLock | Used to prevent concurrent execution with invalid message deletion |
| SInvalWriteLock | Used to prevent concurrent execution with invalid message write and deletion |
| WALInsertLock | Used to prevent concurrent execution with WAL insertion |
| WALWriteLock | Used to prevent concurrent write from a WAL buffer to a disk |
| ControlFileLock | Used to prevent concurrent read/write or concurrent write/write on the pg_control file |
| CheckpointLock | Used to prevent multi-checkpoint concurrent execution |
| CLogControlLock | Used to prevent concurrent access to or concurrent modification on the Clog control data structure |
| MultiXactGenLock | Used to allocate a unique MultiXact ID in serial mode |
| MultiXactOffsetControl-Lock | Used to prevent concurrent read/write or concurrent write/write on pg_multixact/offset |
| MultiXactMemberControl-Lock | Used to prevent concurrent read/write or concurrent write/write on pg_multixact/members |
| RelCacheInitLock | Used to add a lock before any operations are performed on the init file when messages are invalid |
| CheckpointInterCommLock | Used to send file flush requests to a checkpoint. The request structure needs to be inserted to a request queue in serial mode. |
| TwoPhaseStateLock | Used to prevent concurrent access to or modification on two-phase information sharing arrays |
| TablespaceCreateLock | Used to check whether a tablespace already exists |
| BtreeVacuumLock | Used to prevent VACUUM from clearing pages that are being used by B-tree indexes |
| AutovacuumLock | Used to access the autovacuum worker array in serial mode |
| AutovacuumScheduleLock | Used to distribute tables requiring VACUUM in serial mode |

| wait_event | Description |
|------------------------------------|---|
| SyncScanLock | Used to determine the start position of a relfilenode during heap scanning |
| NodeTableLock | Used to protect a shared structure that stores CN and DN information |
| PoolerLock | Used to prevent two threads from simultaneously obtaining the same connection from a connection pool |
| RelationMappingLock | Used to wait for the mapping file between system catalogs and storage locations to be updated |
| AsyncCtlLock | Used to prevent concurrent access to or concurrent modification on the sharing notification status |
| AsyncQueueLock | Used to prevent concurrent access to or concurrent modification on the sharing notification queue |
| SerializableXactHashLock | Used to prevent concurrent read/write or concurrent write/write on a sharing structure for serializable transactions |
| SerializableFinishedList-Lock | Used to prevent concurrent read/write or concurrent write/write on a shared linked list for completed serial transactions |
| SerializablePredicateLock-ListLock | Used to protect a linked list of serializable transactions that have locks |
| OldSerXidLock | Used to protect a structure that records serializable transactions that have conflicts |
| FileStatLock | Used to protect a data structure that stores statistics file information |
| SyncRepLock | Used to protect Xlog synchronization information during primary-standby replication |
| DataSyncRepLock | Used to protect data page synchronization information during primary-standby replication |
| CStoreColspaceCacheLock | Used to add a lock when CU space is allocated for a column-store table |
| CStoreCUCacheSweep-Lock | Used to add a lock when CU caches used by a column-store table are cyclically washed out |
| MetaCacheSweepLock | Used to add a lock when metadata is cyclically washed out |
| DfsConnectorCacheLock | Used to protect a global hash table where HDFS connection handles are cached |

| wait_event | Description |
|--------------------------------|--|
| dummyServerInfoCache-Lock | Used to protect a global hash table where the information about computing Node Group connections is cached |
| ExtensionConnectorLibLock | Used to add a lock when a specific dynamic library is loaded or uninstalled in ODBC connection initialization scenarios |
| SearchServerLibLock | Used to add a lock on the file read operation when a specific dynamic library is initially loaded in GPU-accelerated scenarios |
| DfsUserLoginLock | Used to protect a global linked table where HDFS user information is stored |
| DfsSpaceCacheLock | Used to ensure that the IDs of files to be imported to an HDFS table increase monotonically |
| LsnXlogChkFileLock | Used to serially update the Xlog flush points for primary and standby servers recorded in a specific structure |
| GTMHostInfoLock | Used to prevent concurrent access to or concurrent modification on GTM host information |
| ReplicationSlotAllocation-Lock | Used to add a lock when a primary server allocates stream replication slots during primary-standby replication |
| ReplicationSlotControl-Lock | Used to prevent concurrent update of replication slot status during primary-standby replication |
| ResourcePoolHashLock | Used to prevent concurrent access to or concurrent modification on a resource pool table, a hash table |
| WorkloadStatHashLock | Used to prevent concurrent access to or concurrent modification on a hash table that contains SQL requests from the CN side |
| WorkloadIoStatHashLock | Used to prevent concurrent access to or concurrent modification on a hash table that contains the I/O information of the current DN |
| WorkloadCGroupHash-Lock | Used to prevent concurrent access to or concurrent modification on a hash table that contains Cgroup information |
| OBSGetPathLock | Used to prevent concurrent read/write or concurrent write/write on an OBS path |
| WorkloadUserInfoLock | Used to prevent concurrent access to or concurrent modification on a hash table that contains user information about load management |

| wait_event | Description |
|------------------------|--|
| WorkloadRecordLock | Used to prevent concurrent access to or concurrent modification on a hash table that contains requests received by CNs during adaptive memory management |
| WorkloadIOUtilLock | Used to protect a structure that records iostat and CPU load information |
| WorkloadNodeGroupLock | Used to prevent concurrent access to or concurrent modification on a hash table that contains Node Group information in memory |
| JobShmemLock | Used to protect global variables in the shared memory that is periodically read during a scheduled task where MPP is compatible with Oracle |
| OBSRuntimeLock | Used to obtain environment variables, for example, <i>GAUSSHOME</i> . |
| LLVMDumpIRLock | Used to export the assembly language for dynamically generating functions |
| LLVMParseIRLock | Used to compile and parse a finished IR function from the IR file at the start position of a query |
| RPNNumberLock | Used by a DN on a computing Node Group to count the number of threads for a task where plans are being executed |
| ClusterRPLock | Used to control concurrent access on cluster load data maintained in a CCN of the cluster |
| CriticalCacheBuildLock | Used to load caches from a shared or local cache initialization file |
| WaitCountHashLock | Used to protect a shared structure in user statement counting scenarios |
| BufMappingLock | Used to protect operations on a table mapped to shared buffer |
| LockMgrLock | It is used to protect a common lock structure. |
| PredicateLockMgrLock | Used to protect a lock structure that has serializable transactions |
| OperatorRealTLock | Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the operator level |
| OperatorHistLock | Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the operator level |

| wait_event | Description |
|----------------------|--|
| SessionRealTLock | Used to prevent concurrent access to or concurrent modification on a global structure that contains real-time data at the query level |
| SessionHistLock | Used to prevent concurrent access to or concurrent modification on a global structure that contains historical data at the query level |
| CacheSlotMappingLock | Used to protect global CU cache information |
| BarrierLock | Used to ensure that only one thread is creating a barrier at a time |

The following table describes the corresponding wait events when **wait_status** is **wait io**.

Table 18-241 List of wait events corresponding to I/Os

| wait_event | Description |
|------------------------|---|
| BufFileRead | Reads data from a temporary file to a specified buffer. |
| BufFileWrite | Writes the content of a specified buffer to a temporary file. |
| ControlFileRead | Reads the pg_control file, mainly during database startup, checkpoint execution, and primary/standby verification. |
| ControlFileSync | Flushes the pg_control file to a disk, mainly during database initialization. |
| ControlFileSyncUpdate | Flushes the pg_control file to a disk, mainly during database startup, checkpoint execution, and primary/standby verification. |
| ControlFileWrite | Writes to the pg_control file, mainly during database initialization. |
| ControlFileWriteUpdate | Updates the pg_control file, mainly during database startup, checkpoint execution, and primary/standby verification. |
| CopyFileRead | Reads a file during file copying. |
| CopyFileWrite | Writes a file during file copying. |
| DataFileExtend | Writes a file during file extension. |
| DataFileFlush | Flushes a table data file to a disk. |
| DataFileImmediateSync | Flushes a table data file to a disk immediately. |

| wait_event | Description |
|-----------------------------|---|
| DataFilePrefetch | Reads a table data file asynchronously. |
| DataFileRead | Reads a table data file synchronously. |
| DataFileSync | Flushes table data file modifications to a disk. |
| DataFileTruncate | Truncates a table data file. |
| DataFileWrite | Writes a table data file. |
| LockFileAddToDataDir-Read | Reads the postmaster.pid file. |
| LockFileAddToDataDir-Sync | Flushes the postmaster.pid file to a disk. |
| LockFileAddToDataDir-Write | Writes the PID information into the postmaster.pid file. |
| LockFileCreateRead | Read the LockFile file %s.lock . |
| LockFileCreateSync | Flushes the LockFile file %s.lock to a disk. |
| LockFileCreateWRITE | Writes the PID information into the LockFile file %s.lock . |
| RelationMapRead | Reads the mapping file between system catalogs and storage locations. |
| RelationMapSync | Flushes the mapping file between system catalogs and storage locations to a disk. |
| RelationMapWrite | Writes the mapping file between system catalogs and storage locations. |
| ReplicationSlotRead | Reads a stream replication slot file during a restart. |
| ReplicationSlotRestore-Sync | Flushes a stream replication slot file to a disk during a restart. |
| ReplicationSlotSync | Flushes a temporary stream replication slot file to a disk during checkpoint execution. |
| ReplicationSlotWrite | Writes a temporary stream replication slot file during checkpoint execution. |
| SLRUFlushSync | Flushes the pg_clog , pg_subtrans , and pg_multixact files to a disk, mainly during checkpoint execution and database shutdown. |
| SLRURead | Reads the pg_clog , pg_subtrans , and pg_multixact files. |
| SLRUSync | Writes dirty pages into the pg_clog , pg_subtrans , and pg_multixact files, and flushes the files to a disk, mainly during checkpoint execution and database shutdown. |

| wait_event | Description |
|----------------------|--|
| SLRUWrite | Writes the pg_clog , pg_subtrans , and pg_multixact files. |
| TimelineHistoryRead | Reads the timeline history file during database startup. |
| TimelineHistorySync | Flushes the timeline history file to a disk during database startup. |
| TimelineHistoryWrite | Writes to the timeline history file during database startup. |
| TwophaseFileRead | Reads the pg_twophase file, mainly during two-phase transaction submission and restoration. |
| TwophaseFileSync | Flushes the pg_twophase file to a disk, mainly during two-phase transaction submission and restoration. |
| TwophaseFileWrite | Writes the pg_twophase file, mainly during two-phase transaction submission and restoration. |
| WALBootstrapSync | Flushes an initialized WAL file to a disk during database initialization. |
| WALBootstrapWrite | Writes an initialized WAL file during database initialization. |
| WALCopyRead | Read operation generated when an existing WAL file is read for replication after archiving and restoration. |
| WALCopySync | Flushes a replicated WAL file to a disk after archiving and restoration. |
| WALCopyWrite | Write operation generated when an existing WAL file is read for replication after archiving and restoration. |
| WALInitSync | Flushes a newly initialized WAL file to a disk during log reclaiming or writing. |
| WALInitWrite | Initializes a newly created WAL file to 0 during log reclaiming or writing. |
| WALRead | Reads data from Xlogs during redo operations on two-phase files. |
| WALSyncMethodAssign | Flushes all open WAL files to a disk. |
| WALWrite | Writes a WAL file. |

The following table describes the corresponding wait events when **wait_status** is **acquire lock**.

Table 18-242 List of wait events corresponding to transaction locks

| wait_event | Description |
|------------------|--|
| relation | Adds a lock to a table. |
| extend | Adds a lock to a table being scaled out. |
| partition | Adds a lock to a partitioned table. |
| partition_seq | Adds a lock to a partition of a partitioned table. |
| page | Adds a lock to a table page. |
| tuple | Adds a lock to a tuple on a page. |
| transactionid | Adds a lock to a transaction ID. |
| virtualxid | Adds a lock to a virtual transaction ID. |
| object | Adds a lock to an object. |
| cstore_freespace | Adds a lock to idle column-store space. |
| userlock | Adds a lock to a user. |
| advisory | Adds an advisory lock. |

18.3.168 PG_TABLES

PG_TABLES displays access to each table in the database.

Table 18-243 PG_TABLES columns

| Column | Type | Reference | Description |
|------------|---------|--|---|
| schemaname | Name | PG_NAMESPACE .nspname | Name of the schema that contains the table |
| tablename | Name | PG_CLASS .relname | Name of the table |
| tableowner | Name | pg_get_userbyid(PG_CLASS .relowner) | Owner of the table |
| tablespace | Name | PG_TABLESPACE .spcname | Tablespace that contains the table. The default value is null |
| hasindexes | boolean | PG_CLASS .relhasindex | Whether the table has (or recently had) an index. If it does, its value is true . Otherwise, its value is false . |

| Column | Type | Reference | Description |
|---------------|--------------------------|--|---|
| hasrules | boolean | PG_CLASS .relhasrules | Whether the table has rules. If it does, its value is true . Otherwise, its value is false . |
| hastriggers | boolean | PG_CLASS .RELHASTRIGGERS | Whether the table has triggers. If it does, its value is true . Otherwise, its value is false . |
| tablecreator | Name | pg_get_userbyid(PG_OBJECT .creator) | Table creator. If the creator has been deleted, no value is returned. |
| created | Timestamp with time zone | PG_OBJECT .ctime | Time when the table was created. |
| last_ddl_time | Timestamp with time zone | PG_OBJECT .mtime | Last time when the cluster was modified. |

Example

Query all tables in a specified schema.

```
SELECT tablename FROM PG_TABLES WHERE schemaname = 'myschema';
-----
inventory
product
sales_info
test1
mytable
product_info
customer_info
newproducts
customer_t1
(9 rows)
```

18.3.169 PG_TDE_INFO

PG_TDE_INFO displays the encryption information about the current cluster.

Table 18-244 PG_TDE_INFO columns

| Column | Type | Description |
|------------|------|--|
| is_encrypt | Text | Whether the cluster is an encryption cluster <ul style="list-style-type: none">f: Non-encryption clustert: Encryption cluster |

| Column | Type | Description |
|------------|------|--|
| g_tde_algo | Text | Encryption algorithm <ul style="list-style-type: none">AES-CTR-128 |
| remain | Text | Reserved columns |

Examples

Check whether the current cluster is encrypted, and check the encryption algorithm (if any) used by the current cluster.

```
SELECT * FROM PG_TDE_INFO;  
is_encrypt | g_tde_algo | remain  
-----+-----+-----  
f          | AES-CTR-128 | remain  
(1 row)
```

18.3.170 PG_TIMEZONE_ABBREVS

PG_TIMEZONE_ABBREVS displays all time zone abbreviations that can be recognized by the input routines.

Table 18-245 PG_TIMEZONE_ABBREVS columns

| Column | Type | Description |
|------------|----------|---|
| abbrev | Text | Time zone abbreviation |
| utc_offset | interval | Offset from UTC |
| is_dst | boolean | Whether the abbreviation indicates a daylight saving time (DST) zone. If it does, its value is true . Otherwise, its value is false . |

18.3.171 PG_TIMEZONE_NAMES

PG_TIMEZONE_NAMES displays all time zone names that can be recognized by **SET TIMEZONE**, along with their associated abbreviations, UTC offsets, and daylight saving time statuses.

Table 18-246 PG_TIMEZONE_NAMES columns

| Column | Type | Description |
|------------|----------|-----------------------------|
| Name | Text | Name of the time zone |
| abbrev | Text | Time zone name abbreviation |
| utc_offset | interval | Offset from UTC |

| Column | Type | Description |
|--------|---------|--|
| is_dst | boolean | Whether DST is used. If it is, its value is true . Otherwise, its value is false . |

18.3.172 PG_TOTAL_MEMORY_DETAIL

PG_TOTAL_MEMORY_DETAIL displays the memory usage of a certain node in the database.

Table 18-247 PG_TOTAL_MEMORY_DETAIL columns

| Column | Type | Description |
|----------|------|-------------|
| nodename | Text | Node name |

| Column | Type | Description |
|------------|------|--|
| memorytype | Text | <p>It can be set to any of the following values:</p> <ul style="list-style-type: none">• max_process_memory: memory used by a DWS cluster instance• process_used_memory: memory used by a DWS process• max_dynamic_memory: maximum dynamic memory• dynamic_used_memory: used dynamic memory• dynamic_peak_memory: dynamic peak value of the memory• dynamic_used_shrctx: maximum dynamic shared memory context• dynamic_peak_shrctx: dynamic peak value of the shared memory context• max_shared_memory: maximum shared memory• shared_used_memory: used shared memory• max_cstore_memory: maximum memory allowed for column store• cstore_used_memory: memory used for column store• max_sctpcomm_memory: maximum memory allowed for the communication library• sctpcomm_used_memory: memory used for the communication library• sctpcomm_peak_memory: memory peak of the communication library• max_topsql_memory: maximum memory that can be used by top SQL to record historical job monitoring information• topsql_used_memory: memory used by top SQL to record historical job monitoring information• topsql_peak_memory: memory peak of top SQL to record historical job monitoring information• other_used_memory: other used memory• gpu_max_dynamic_memory: maximum GPU memory |

| Column | Type | Description |
|-------------|---------|---|
| | | <ul style="list-style-type: none">• gpu_dynamic_used_memory: sum of the available GPU memory and temporary GPU memory• gpu_dynamic_peak_memory: maximum memory used for GPU• pooler_conn_memory: memory used for pooler connections• pooler_freeconn_memory: memory used for idle pooler connections• storage_compress_memory: memory used for column-store compression and decompression• udf_reserved_memory: memory reserved for the UDF Worker process• mmap_used_memory: memory used for mmap |
| memorybytes | Integer | Size of the used memory (MB) |

18.3.173 PG_TOTAL_SCHEMA_INFO

PG_TOTAL_SCHEMA_INFO displays the storage usage of all schemas in each database. This view is valid only if use_workload_manager is set to **on**.

| Column | Type | Description |
|--------------|--------|---|
| schemaid | OID | Schema OID |
| schemaname | Text | Schema name |
| databaseid | OID | Database OID |
| databasename | Name | Database name |
| usedspace | Bigint | Size of the permanent table storage space used by the schema, in bytes. |
| permspace | Bigint | Upper limit of the permanent table storage space of the schema, in bytes. |

18.3.174 PG_TOTAL_USER_RESOURCE_INFO

PG_TOTAL_USER_RESOURCE_INFO displays the resource usage of all users. Only administrators can query this view. This view is valid only if `use_workload_manager` is set to **on**.

Table 18-248 PG_TOTAL_USER_RESOURCE_INFO columns

| Column | Type | Description |
|------------------|------------------|--|
| username | Name | Username |
| used_memory | Integer | Memory used by a user, in MB. <ul style="list-style-type: none">On a DN, it indicates the memory used by users on the current DN.On a CN, it indicates the total memory used by users on all DNs. |
| total_memory | Integer | Memory used by the resource pool, in MB. 0 indicates that the maximum available memory is not limited and depends on the maximum available memory of the database (max_dynamic_memory). The calculation formula is as follows:
$\text{total_memory} = \text{max_dynamic_memory} * \text{parent_percent} * \text{user_percent}$
On a CN, it indicates the total maximum available memory on all DNs. |
| used_cpu | Double precision | Number of CPU cores in use. Only the CPU usage of complex jobs in the non-default resource pool is collected, and the value is the CPU usage of the related cgroup. |
| total_cpu | Integer | Total number of CPU cores of the Cgroup associated with a user on the node |
| used_space | Bigint | Used permanent table storage space (unit: KB) |
| total_space | Bigint | Available storage space (unit: KB). -1 indicates that the storage space is not limited. |
| used_temp_space | Bigint | Used temporary table storage space (unit: KB) |
| total_temp_space | Bigint | Available temporary table storage space (unit: KB). -1 indicates that the storage space is not limited. |
| used_spill_space | Bigint | Size of the used operator flushing space, in KB |

| Column | Type | Description |
|-------------------|------------------|--|
| total_spill_space | Bigint | Size of the available operator flushing space, in KB. The value -1 indicates that the operator flushing space is not limited. |
| read_kbytes | Bigint | On a CN, it indicates the total number of bytes logically read by a user on all DNs in the last 5 seconds, in KB.
On a DN, it indicates the total number of bytes logically read by a user from the instance startup time to the current time, in KB. |
| write_kbytes | Bigint | On a CN, it indicates the total number of bytes logically written by a user on all DNs in the last 5 seconds, in KB.
On a DN, it indicates the total number of bytes logically written by a user from the instance startup time to the current time, in KB. |
| read_counts | Bigint | On a CN, it indicates the total number of logical reads performed by a user on all DNs in the last 5 seconds.
On a DN, it indicates the total number of logical reads performed by a user from the instance startup time to the current time. |
| write_counts | Bigint | On a CN, it indicates the total number of logical writes performed by a user on all DNs in the last 5 seconds.
On a DN, it indicates the total number of logical writes performed by a user from the instance startup time to the current time. |
| read_speed | Double precision | On a CN, it indicates the sum of average logical read rates of a user on all DNs in the last 5 seconds, in KB/s.
On a DN, it indicates the average logical read rate of a user on the DN in the last 5 seconds, in KB/s. |
| write_speed | Double precision | On a CN, it indicates the sum of average logical write rates of a user on all DNs in the last 5 seconds, in KB/s.
On a DN, it indicates the average logical write rate of a user on the DN in the last 5 seconds, in KB/s. |

| Column | Type | Description |
|------------|------------------|---|
| send_speed | Double precision | On a CN, it indicates the sum of the average network sending rates of a user on all DNs in the last 5 seconds, in KB/s.
On a DN, it indicates the average network sending rate of a user on the DN in the last 5 seconds, in KB/s. |
| recv_speed | Double precision | On a CN, it indicates the sum of the average network receiving rates of a user on all DNs in the last 5 seconds, in KB/s.
On a DN, it indicates the average network receiving rate of a user on the DN in the last 5 seconds, in KB/s. |

18.3.175 PG_USER

PG_USER displays information about users who can access the database.

Table 18-249 PG_USER columns

| Column | Type | Description |
|-------------|--------------------------|--|
| username | Name | User name |
| usesysid | OID | ID of this user |
| usecreatedb | boolean | Whether the user has the permission to create databases |
| usesuper | boolean | whether the user is the initial system administrator with the highest rights. |
| usecatupd | boolean | whether the user can directly update system tables. Only the initial system administrator whose usesysid is 10 has this permission. It is not available for other users. |
| userepl | boolean | Whether the user has the permission to duplicate data streams |
| passwd | Text | Encrypted user password. The value is displayed as *****. |
| valbegin | Timestamp with time zone | Account validity start time; null if no start time |
| valuntil | Timestamp with time zone | Password expiry time; null if no expiration |

| Column | Type | Description |
|-----------------|--------|---|
| respool | Name | Resource pool where the user is in |
| parent | OID | Parent user OID |
| spacelimit | Text | The storage space of the permanent table. |
| tempspacelimit | Text | The storage space of the temporary table. |
| spillspacelimit | Text | The operator disk flushing space. |
| useconfig | Text[] | Session defaults for run-time configuration variables |
| nodegroup | Name | Name of the logical cluster associated with the user. If no logical cluster is associated, this column is left blank. |

Example

Query the current database user list.

```
SELECT username FROM pg_user;  
username  
-----  
dbadmin  
u1  
u2  
u3  
(4 rows)
```

18.3.176 PG_USER_MAPPINGS

PG_USER_MAPPINGS displays information about user mappings.

This is essentially a publicly readable view of **PG_USER_MAPPING** that leaves out the options column if the user has no rights to use it.

Table 18-250 PG_USER_MAPPINGS columns

| Column | Type | Reference | Description |
|---------|------|---|---|
| umid | OID | PG_USER_MAPPING.oid | OID of the user mapping |
| srvid | OID | PG_FOREIGN_SERVER.oid | OID of the foreign server that contains this mapping |
| srvname | Name | PG_FOREIGN_SERVER.srvname | Name of the foreign server |
| umuser | OID | PG_AUTHID.oid | OID of the local role being mapped, 0 if the user mapping is public |

| Column | Type | Reference | Description |
|-----------|--------|-----------|--|
| username | Name | - | Name of the local user to be mapped |
| umoptions | text[] | - | User mapping specific options. If the current user is the owner of the foreign server, its value is keyword=value strings. Otherwise, its value is null. |

18.3.177 PG_VIEWS

PG_VIEWS displays basic information about each view in the database.

Table 18-251 PG_VIEWS columns

| Column | Type | Reference | Description |
|------------|------|---------------------------------------|---|
| schemaname | Name | PG_NAMESPACE .nspname | Name of the schema that contains the view |
| viewname | Name | PG_CLASS .relname | View name |
| viewowner | Name | PG_AUTHID .rolname | Owner of the view |
| definition | Text | - | Definition of the view |

Example

Query all the views in a specified schema.

```
SELECT * FROM pg_views WHERE schemaname = 'myschema';
schemaname | viewname | viewowner | definition
-----+-----+-----+-----
myschema   | myview   | dbadmin   | SELECT * FROM pg_tablespace WHERE (pg_tablespace.spcname =
pg_default::name);
myschema   | v1       | dbadmin   | SELECT * FROM t1 WHERE (t1.c1 > 200);
(2 rows)
```

18.3.178 PG_WLM_STATISTICS

PG_WLM_STATISTICS displays information about workload management after the task is complete or the exception has been handled. This view has been discarded in 8.1.2. You can use [PGXC_WLM_SESSION_INFO](#) to view load management records of completed jobs executed on all CNs.

Table 18-252 PG_WLM_STATISTICS columns

| Column | Type | Description |
|--------------------|---------|--|
| statement | Text | Statement executed for exception handling |
| block_time | Bigint | Block time before the statement is executed |
| elapsed_time | Bigint | Elapsed time when the statement is executed |
| total_cpu_time | Bigint | Total time used by the CPU on the DN when the statement is executed for exception handling |
| qualification_time | Bigint | Period when the statement checks the inclination ratio |
| cpu_skew_percent | Integer | CPU usage skew on the DN when the statement is executed for exception handling |
| control_group | Text | Cgroup used when the statement is executed for exception handling |
| status | Text | Statement status after it is executed for exception handling <ul style="list-style-type: none">• pending: The statement is waiting to be executed.• running: The statement is being executed.• finished: The execution is finished normally.• abort: The execution is unexpectedly terminated. |
| action | Text | Actions when statements are executed for exception handling <ul style="list-style-type: none">• abort indicates terminating the operation.• adjust indicates executing the Cgroup adjustment operations. Currently, you can only perform the demotion operation.• finish indicates that the operation is normally finished. |
| queryid | Bigint | Internal query ID used for statement execution |
| threadid | Bigint | ID of the backend thread |

18.3.179 PGXC_BULKLOAD_PROGRESS

PGXC_BULKLOAD_PROGRESS displays the progress of the service import. Only GDS common files can be imported. This view is accessible only to users with system administrators rights.

Table 18-253 PGXC_BULKLOAD_PROGRESS columns

| Column | Type | Description |
|------------|--------|---|
| session_id | Bigint | GDS session ID |
| query_id | Bigint | Query ID. It is equivalent to debug_query_id . |
| query | Text | Query statement |
| progress | Text | Progress percentage |

18.3.180 PGXC_BULKLOAD_INFO

By querying the **PGXC_BULKLOAD_INFO** view on CNs, you can obtain historical statistics information for interconnection, GDS, COPY, and \COPY business executions after they have completed. This view summarizes the historical execution information of import and export business that have already completed on each node of the current cluster (including the interconnection cluster address, import and export business type, maximum, minimum, and total number of rows and bytes written to disk on DNs, etc.), to obtain historical information on import and export business execution and assist in performance troubleshooting.

This view does not record abnormal interruptions of import and export jobs. The data is directly obtained from the system catalog **GS_WLM_SESSION_INFO**, and the **loader_status** field is parsed to obtain import and export service information.

System administrator rights are required to access this view.

Table 18-254 PGXC_BULKLOAD_INFO columns

| Column | Type | Description |
|------------------|------|---|
| datid | OID | OID of the database the backend is connected to. |
| dbname | Text | Name of the database the backend is connected to. |
| schemaname | Text | Schema name. |
| nodename | Text | Name of the CN where the statement is run. |
| username | Text | Username for connecting to the backend. |
| application_name | Text | Name of the application that is connected to the backend. |

| Column | Type | Description |
|-----------------|--------------------------|---|
| client_addr | inet | IP address of the client connected to the backend. If this column is null, it indicates that the client is connected via a Unix socket on the server machine or that it is an internal process, such as autovacuum. |
| client_hostname | Text | Host name of the client, which is obtained by reverse DNS lookup of client_addr . This column is only non-null when log_hostname is enabled and IP connection is used. |
| client_port | Integer | TCP port number used by the client to communicate with the backend. If a Unix socket is used, it is -1 . |
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |
| block_time | Bigint | Blocking time before statement execution, including statement parsing and optimization time, in milliseconds. |
| start_time | Timestamp with time zone | Start time of statement execution. |
| finish_time | Timestamp with time zone | End time of statement execution. |
| status | Text | End status of statement execution: finished for normal and aborted for abnormal. The statement status recorded here should be the database server execution status. When the server-side execution is successful and an error occurs when the result set is returned, the statement should be finished . |
| queryid | Bigint | Internal query ID used for statement execution. |
| query | Text | Executed statement. |
| session_id | Text | A session uniquely identified in the database system, in the format of session_start_time.tid.node_name . |
| address | Text | Server address of the interconnection peer cluster. When not empty, it indicates an interconnection service, and the source cluster will additionally obtain the remote cluster port number. |

| Column | Type | Description |
|------------------|------|--|
| direction | Text | Type of import and export service, including gds to file , gds from file , gds to pipe , gds from pipe , copy from , and copy to . |
| min_done_lines | json | Minimum number of rows of a statement across all DNs. |
| max_done_lines | json | Maximum number of rows of a statement across all DNs. |
| total_done_lines | json | Total number of rows of a statement across all DNs. |
| min_done_bytes | json | Minimum number of bytes of a statement across all DNs. |
| max_done_bytes | json | Maximum number of bytes of a statement across all DNs. |
| total_done_bytes | json | Total number of bytes of a statement across all DNs. |

 NOTE

- Abnormal interruptions of import and export jobs are not recorded in the view.
- The implementation mechanism of GDS foreign tables and interconnection foreign tables is different. When querying, GDS records the full amount, while interconnection records the actual amount.
- For non-full import and export foreign tables with a limit, due to the special execution plan of limit, the data displayed is collected from one DN, which appears as a maximum value of all and a minimum value of 0.
- If the import and export table is a non-partitioned table:
 - When the GDS partitioned table is small, if one DN has finished collecting data and the other DNs have not started collecting data, they will not collect data. Therefore, when the data volume of GDS from non-partitioned tables is small, the minimum value may be 0, but it is not 0 when the table data volume is large.
 - When exporting non-partitioned tables from the interconnection source cluster, all DNs will be recorded, and only one DN's data will be collected, so the minimum value is 0.
 - When exporting replication tables from the interconnection remote cluster, only one DN will be recorded, so it is equivalent to having only one DN, and the minimum and maximum values are the same.
- Historical monitoring of import and export is implemented by reusing the historical TopSQL function, which follows the precautions, prerequisites, and operation steps of TopSQL. For details, refer to [Historical Top SQL](#).
- Due to the large amount of data recorded by TopSQL, you are advised to query and use it as needed by combining fields such as **start_time** and **finish_time** to improve query performance, or to reduce query frequency.

| Name | Type | Description |
|-------------|--------------------------|--|
| db_name | Text | Database name |
| query_id | Bigint | Query ID. It is equivalent to debug_query_id . |
| tid | Bigint | ID of the current thread |
| lwtid | Integer | Lightweight thread ID |
| session_id | Bigint | GDS session ID |
| direction | Text | Service type. The options are gds to file , gds from file , gds to pipe , gds from pipe , copy from , and copy to . |
| query | Text | Query statement |
| address | Text | Location of the foreign table used for data import and export |
| query_start | Timestamp with time zone | Start time of data import or export |
| total_bytes | Bigint | Total size of data to be processed
This parameter is specified only when a GDS common file is to be imported and the record in the row comes from a CN. Otherwise, left this parameter unspecified. |
| phase | Text | Current phase. The options are INITIALIZING , TRANSFER_DATA , and RELEASE_RESOURCE . |
| done_lines | Bigint | Number of lines that have been transferred |
| done_bytes | Bigint | Number of bytes that have been transferred |

18.3.182 PGXC_COLUMN_TABLE_IO_STAT

PGXC_COLUMN_TABLE_IO_STAT provides I/O statistics of all column-store tables of the database on all CNs and DN in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_COLUMN_TABLE_IO_STAT** view. For details about the columns, see [Table 18-256](#).

Table 18-256 GS_COLUMN_TABLE_IO_STAT columns

| Column | Type | Description |
|------------|--------|--|
| schemaname | Name | Namespace of a table |
| relname | Name | Table name |
| heap_read | Bigint | Number of blocks logically read in the heap |
| heap_hit | Bigint | Number of block hits in the heap |
| idx_read | Bigint | Number of blocks logically read in the index |
| idx_hit | Bigint | Number of block hits in the index |
| cu_read | Bigint | Number of logical reads in the Compression Unit |
| cu_hit | Bigint | Number of hits in the Compression Unit |
| cidx_read | Bigint | Number of indexes logically read in the Compression Unit |
| cidx_hit | Bigint | Number of index hits in the Compression Unit |

18.3.183 PGXC_COMM_CLIENT_INFO

PGXC_COMM_CLIENT_INFO stores the client connection information of all nodes. (You can query this view on a DN to view the information about the connection between the CN and DN.)

Table 18-257 PGXC_COMM_CLIENT_INFO columns

| Column | Type | Description |
|-------------|---------|---|
| node_name | Text | Current node name. |
| app | Text | Client application name. |
| tid | Bigint | Thread ID of the current thread. |
| lwtid | Integer | Lightweight thread ID of the current thread. |
| query_id | Bigint | Query ID. It is equivalent to debug_query_id . |
| socket | Integer | It is displayed if the connection is a physical connection. |
| remote_ip | Text | Peer node IP address. |
| remote_port | Text | Peer node port. |

| Column | Type | Description |
|----------|---------|---|
| logic_id | Integer | If the connection is a logical connection, sid is displayed. If -1 is displayed, the current connection is a physical connection. |

18.3.184 PGXC_COMM_DELAY

PGXC_COMM_DELAY displays the communication library delay status for all the DNs.

Table 18-258 PGXC_COMM_DELAY columns

| Column | Type | Description |
|-------------|---------|---|
| node_name | Text | Node name |
| remote_name | Text | Name of the peer node with the maximum connection latency. |
| remote_host | Text | IP address of the peer |
| stream_num | Integer | Number of logical stream connections used by the current physical connection |
| min_delay | Integer | Minimum delay of the current physical connection. The unit is microsecond. |
| average | Integer | Average delay of the current physical connection. The unit is microsecond. |
| max_delay | Integer | Maximum delay of the current physical connection. The unit is microsecond.
NOTE
If its value is -1 , the latency detection has timed out. In this case, re-establish the connection between nodes and then perform the query. |

18.3.185 PGXC_COMM_RECV_STREAM

PG_COMM_RECV_STREAM displays the receiving stream status of the communication libraries for all the DNs.

Table 18-259 PGXC_COMM_RECV_STREAM columns

| Column | Type | Description |
|-----------|--------|------------------------------------|
| node_name | Text | Node name |
| local_tid | Bigint | ID of the thread using this stream |

| Column | Type | Description |
|-------------|---------|---|
| remote_name | Text | Name of the peer node |
| remote_tid | Bigint | Peer thread ID |
| idx | Integer | Peer DN ID in the local DN |
| sid | Integer | Stream ID in the physical connection |
| tcp_sock | Integer | TCP socket used in the stream |
| state | Text | Current status of the stream <ul style="list-style-type: none">• UNKNOWN: The logical connection is unknown.• READY: The logical connection is ready.• RUN: The logical connection receives packets normally.• HOLD: The logical connection is waiting to receive packets.• CLOSED: The logical connection is closed.• TO_CLOSED: The logical connection is to be closed.• WRITING: Data is being written. |
| query_id | Bigint | debug_query_id corresponding to the stream |
| pn_id | Integer | plan_node_id of the query executed by the stream |
| send_smp | Integer | smpid of the sender of the query executed by the stream |
| recv_smp | Integer | smpid of the receiver of the query executed by the stream |
| recv_bytes | Bigint | Total data volume received from the stream. The unit is byte. |
| time | Bigint | Current life cycle service duration of the stream. The unit is ms. |
| speed | Bigint | Average receiving rate of the stream. The unit is byte/s. |
| quota | Bigint | Current communication quota value of the stream. The unit is Byte. |
| buff_usize | Bigint | Current size of the data cache of the stream. The unit is byte. |

18.3.186 PGXC_COMM_SEND_STREAM

PGXC_COMM_SEND_STREAM displays the sending stream status of the communication libraries for all the DNs.

Table 18-260 PGXC_COMM_SEND_STREAM columns

| Column | Type | Description |
|-------------|---------|--|
| node_name | Text | Node name |
| local_tid | Bigint | ID of the thread using this stream |
| remote_name | Text | Name of the peer node |
| remote_tid | Bigint | Peer thread ID |
| idx | Integer | Peer DN ID in the local DN |
| sid | Integer | Stream ID in the physical connection |
| tcp_sock | Integer | TCP socket used in the stream |
| state | Text | Current status of the stream. <ul style="list-style-type: none">• UNKNOWN: The logical connection is unknown.• READY: The logical connection is ready.• RUN: The logical connection sends packets normally.• HOLD: The logical connection is waiting to send packets.• CLOSED: The logical connection is closed.• TO_CLOSED: The logical connection is to be closed.• WRITING: Data is being written. |
| query_id | Bigint | debug_query_id corresponding to the stream |
| pn_id | Integer | plan_node_id of the query executed by the stream |
| send_smp | Integer | smpid of the sender of the query executed by the stream |
| recv_smp | Integer | smpid of the receiver of the query executed by the stream |
| send_bytes | Bigint | Total data volume sent by the stream. The unit is Byte. |
| time | Bigint | Current life cycle service duration of the stream. The unit is ms. |
| speed | Bigint | Average sending rate of the stream. The unit is Byte/s. |

| Column | Type | Description |
|------------|--------|---|
| quota | Bigint | Current communication quota value of the stream. The unit is Byte. |
| wait_quota | Bigint | Extra time generated when the stream waits the quota value. The unit is ms. |

18.3.187 PGXC_COMM_STATUS

PGXC_COMM_STATUS displays the communication library status for all the DNs.

Table 18-261 PGXC_COMM_STATUS columns

| Column | Type | Description |
|----------------|---------|--|
| node_name | Text | Node name. |
| rxpck/s | Integer | Receiving rate of the communication library on a node. The unit is byte/s. |
| txpck/s | Integer | Sending rate of the communication library on a node. The unit is byte/s. |
| rxkB/s | Bigint | Receiving rate of the communication library on a node. The unit is KB/s. |
| txkB/s | Bigint | Sending rate of the communication library on a node. The unit is KB/s. |
| buffer | Bigint | Size of the buffer of the Cmailbox. |
| memKB(libcomm) | Bigint | Communication memory size of the libcomm process, in KB. |
| memKB(libpq) | Bigint | Communication memory size of the libpq process, in KB. |
| %USED(PM) | Integer | Real-time usage of the postmaster thread. |
| %USED (sflow) | Integer | Real-time usage of the gs_sender_flow_controller thread. |
| %USED (rflow) | Integer | Real-time usage of the gs_receiver_flow_controller thread. |
| %USED (rloop) | Integer | Highest real-time usage among multiple gs_receivers_loop threads. |
| stream | Integer | Total number of used logical connections. |

18.3.188 PGXC_COMM_QUERY_SPEED

PGXC_COMM_QUERY_SPEED displays traffic information about all queries on all nodes.

Table 18-262 PGXC_COMM_QUERY_SPEED columns

| Column | Type | Description |
|-----------|--------|--|
| node_name | Text | Node name |
| query_id | Bigint | debug_query_id corresponding to the stream |
| rxkB/s | Bigint | Receiving rate of the query stream (unit: byte/s) |
| txkB/s | Bigint | Sending rate of the query stream (unit: byte/s) |
| rxkB | Bigint | Total received data of the query stream (unit: byte) |
| txkB | Bigint | Total sent data of the query stream (unit: byte) |
| rxpck/s | Bigint | Packet receiving rate of the query (unit: packets/s) |
| txpck/s | Bigint | Packet sending rate of the query (Unit: packets/s) |
| rxpck | Bigint | Total number of received packets of the query |
| txpck | Bigint | Total number of sent packets of the query |

18.3.189 PGXC_DEADLOCK

PGXC_DEADLOCK displays lock wait information generated due to distributed deadlocks.

Currently, **PGXC_DEADLOCK** collects only lock wait information about locks whose **locktype** is **relation**, **partition**, **page**, **tuple**, or **transactionid**.

Table 18-263 PGXC_DEADLOCK columns

| Column | Type | Description |
|----------|------|--|
| locktype | Text | Type of the locked object |
| nodename | Name | Name of the node where the locked object resides |

| Column | Type | Description |
|---------------|--------------------------|--|
| dbname | Name | Name of the database where the locked object resides. The value is NULL if the locked object is a transaction. |
| nspname | Name | Name of the namespace of the locked object |
| relname | Name | Name of the relation targeted by the lock. The value is NULL if the object is not a relation or part of a relation. |
| partname | Name | Name of the partition targeted by the lock. The value is NULL if the locked object is not a partition. |
| page | Integer | Number of the page targeted by the lock. The value is NULL if the locked object is neither a page nor a tuple. |
| tuple | Smallint | Number of the tuple targeted by the lock. The value is NULL if the locked object is not a tuple. |
| transactionid | Xid | ID of the transaction targeted by the lock. The value is NULL if the locked object is not a transaction. |
| waitusername | Name | Name of the user who waits for the lock |
| waitgxid | Xid | ID of the transaction that waits for the lock |
| waitxactstart | Timestamp with time zone | Start time of the transaction that waits for the lock |
| waitqueryid | Bigint | Latest query ID of the thread that waits for the lock |
| waitquery | Text | Latest query statement of the thread that waits for the lock |
| waitpid | Bigint | ID of the thread that waits for the lock |
| waitmode | Text | Mode of the waited lock |
| holdusername | Name | Name of the user who holds the lock |
| holdgxid | Xid | ID of the transaction that holds the lock |
| holdxactstart | Timestamp with time zone | Start time of the transaction that holds the lock |
| holdqueryid | Bigint | Latest query ID of the thread that holds the lock |

| Column | Type | Description |
|-----------|--------------------------|--|
| holdquery | Text | Latest query statement of the thread that holds the lock |
| holdpid | Bigint | ID of the thread that holds the lock |
| holdmode | Text | Mode of the held lock |
| waittime | Timestamp with time zone | Timestamp when the lock wait starts.
This column is available only in clusters of version 9.1.0.200 or later. |
| holdtime | Timestamp with time zone | Timestamp when the lock starts to be held.
This column is available only in clusters of version 9.1.0.200 or later. |

18.3.190 PGXC_DISK_CACHE_STATS

PGXC_DISK_CACHE_STATS records the usage of file cache. This system view is supported only by clusters of version 9.1.0 or later.

Table 18-264 PGXC_DISK_CACHE_STATS columns

| Column | Type | Description |
|-------------|--------------|---|
| node_name | Text | Node name. |
| total_read | Bigint | Total number of accesses to disk cache. |
| local_read | Bigint | Total number of times disk cache reads from local disk. |
| remote_read | Bigint | Total number of times disk cache reads from remote storage. |
| hit_rate | numeric(5,2) | Hit rate of disk cache. |
| cache_size | Bigint | Total size of data saved in disk cache, in KB. |
| fill_rate | numeric(5,2) | Fill rate of disk cache. |

Example

Query the hit rate of disk cache on each node.

```
SELECT hit_rate FROM pgxc_disk_cache_stats;  
hit_rate  
-----
```

| |
|----------|
| 56.91 |
| 56.85 |
| NaN |
| NaN |
| NaN |
| NaN |
| (6 rows) |

18.3.191 PGXC_DISK_CACHE_ALL_STATS

PGXC_DISK_CACHE_ALL_STATS records all usage of file cache. This system view is supported only by clusters of version 9.1.0 or later.

Table 18-265 PGXC_DISK_CACHE_ALL_STATS columns

| Column | Type | Description |
|-----------------|--------------|--|
| node_name | Text | Node name. |
| total_read | Bigint | Total number of accesses to disk cache. |
| local_read | Bigint | Total number of times disk cache accesses local disk. |
| remote_read | Bigint | Total number of times disk cache accesses remote storage. |
| hit_rate | numeric(5,2) | Hit rate of disk cache. |
| cache_size | Bigint | Total size of data saved in disk cache, in KB. |
| fill_rate | numeric(5,2) | Fill rate of disk cache. |
| temp_file_size | Bigint | Total size of temporary/cold cache files, in KB. |
| a1in_size | Bigint | Total size of data saved in the a1in queue of disk cache, in KB. |
| a1out_size | Bigint | Total size of data saved in the a1out queue of disk cache, in KB. |
| am_size | Bigint | Total size of data saved in the am queue of disk cache, in KB. |
| a1in_fill_rate | numeric(5,2) | Fill rate of the a1in queue in disk cache. |
| a1out_fill_rate | numeric(5,2) | Fill rate of the a1out queue in disk cache. |
| am_fill_rate | numeric(5,2) | Fill rate of the am queue in disk cache. |
| fd | Integer | Number of file descriptors currently in use by disk cache. |

| Column | Type | Description |
|-----------------|--------|--|
| pin_block_count | Bigint | Number of pinned blocks in disk cache. This field is supported only by 9.1.0.100 and later cluster versions. |

Example

Query the number of file descriptors used by disk cache on each node.

```
SELECT fd FROM pgxc_disk_cache_all_stats;  
fd  
-----  
1000  
1000  
0  
0  
0  
0  
(6 rows)
```

18.3.192 PGXC_DISK_CACHE_PATH_INFO

PGXC_DISK_CACHE_PATH_INFO records information about the hard disk where the file cache is stored. This system view is supported only by clusters of version 9.1.0 or later.

Table 18-266 PGXC_DISK_CACHE_PATH_INFO columns

| Column | Type | Description |
|----------------|------------------|---|
| path_name | Text | Path name. |
| node_name | Text | Name of the node the hard disk belongs to. |
| cache_size | Bigint | Total size of cache files in the hard disk, in bytes. |
| disk_available | Bigint | Available space in the hard disk, in bytes. |
| disk_size | Bigint | Total capacity of the hard drive, in bytes. |
| disk_use_ratio | Double precision | Disk space usage. |

Example

Query information about the hard disk used by the file cache.

```
SELECT * FROM pgxc_disk_cache_path_info order by 1;  
path_name | node_name | cache_size | disk_available | disk_size | disk_use_ratio  
-----+-----+-----+-----+-----+-----  
dn_6001_6002_0 | dn_6001_6002 | 19619 | 137401716736 | 160982630400 | .146481105479564
```

| | | | | | |
|----------------|--------------|-------|--------------|--------------|------------------|
| dn_6001_6002_1 | dn_6001_6002 | 35968 | 137401716736 | 160982630400 | .146481105479564 |
| dn_6003_6004_0 | dn_6003_6004 | 27794 | 121600655360 | 160982630400 | .244634933235629 |
| dn_6003_6004_1 | dn_6003_6004 | 26158 | 121600655360 | 160982630400 | .244634933235629 |
| dn_6005_6006_0 | dn_6005_6006 | 24533 | 134394839040 | 160982630400 | .165159379579873 |
| dn_6005_6006_1 | dn_6005_6006 | 31065 | 134394839040 | 160982630400 | .165159379579873 |

18.3.193 PGXC_GET_STAT_ALL_TABLES

PGXC_GET_STAT_ALL_TABLES displays information about insertion, update, and deletion operations on tables and the dirty page rate of tables.

Before running **VACUUM FULL** on a system catalog with a high dirty page rate, ensure that no user is performing operations on it.

You are advised to run **VACUUM FULL** to tables (excluding system catalogs) whose dirty page rate exceeds 80% or run it based on service scenarios.

NOTE

For clusters of 8.2.0.100 or later, **PGXC_STAT_TABLE_DIRTY** is recommended for querying the dirty page rate.

Table 18-267 PGXC_GET_STAT_ALL_TABLES columns

| Column | Type | Description |
|-----------------|-------------------|--------------------------------|
| relid | OID | Table OID |
| relname | Name | Table name |
| schemaname | Name | Schema name of the table |
| n_tup_ins | Numeric | Number of inserted tuples |
| n_tup_upd | Numeric | Number of updated tuples |
| n_tup_del | Numeric | Number of deleted tuples |
| n_live_tup | Numeric | Number of live tuples |
| n_dead_tup | Numeric | Number of dead tuples |
| dirty_page_rate | numeric(5,2)
) | Dirty page rate (%) of a table |

For details, see "Functions and Operators" > "System Administration Functions" > "Other Functions" in the *Data Warehouse Service (DWS) SQL Syntax*.

Examples

Use the view **PGXC_GET_STAT_ALL_TABLES** to query the tables whose dirty page rate is greater than 30%.

```
SELECT * FROM PGXC_GET_STAT_ALL_TABLES WHERE dirty_page_rate>30;
relid |      relname      | schemaname | n_tup_ins | n_tup_upd | n_tup_del | n_live_tup | n_dead_tup |
dirty_page_rate
-----+-----+-----+-----+-----+-----+-----+-----+
+-----+
```

```

2840 | pg_toast_2619 | pg_toast | 7415 | 0 | 7415 | 0 | 291 | 88.00
9001 | pgxc_class | pg_catalog | 56331 | 3 | 56285 | 54 | 143 | 72.59
53860 | reason | dbadmin | 9 | 19 | 0 | 9 | 19 | 67.86
9025 | pg_object | pg_catalog | 112858 | 1179707 | 112619 | 246 | 429 |
63.56
9015 | pgxc_node | pg_catalog | 15 | 24 | 0 | 15 | 24 | 61.54
2606 | pg_constraint | pg_catalog | 78 | 0 | 42 | 36 | 42 | 53.85
1260 | pg_authid | pg_catalog | 6 | 6 | 0 | 6 | 6 | 50.00
(7 rows)

```

18.3.194 PGXC_GET_STAT_ALL_PARTITIONS

PGXC_GET_STAT_ALL_PARTITIONS displays information about insertion, update, and deletion operations on partitions of partitioned tables and the dirty page rate of tables.

The statistics of this view depend on the **ANALYZE** operation. To obtain the most accurate information, perform the **ANALYZE** operation on the partitioned table first.

NOTE

For clusters of 8.2.0.100 or later, **PGXC_STAT_TABLE_DIRTY** is recommended for querying the dirty page rate.

Table 18-268 PGXC_GET_STAT_ALL_PARTITIONS columns

| Column | Type | Description |
|-----------------|--------------|--------------------------------|
| relid | OID | Table OID |
| partid | OID | Partition OID |
| schemaname | Name | Schema name of the table |
| relname | Name | Table name |
| partname | Name | Partition name |
| n_tup_ins | Numeric | Number of inserted tuples |
| n_tup_upd | Numeric | Number of updated tuples |
| n_tup_del | Numeric | Number of deleted tuples |
| n_live_tup | Numeric | Number of live tuples |
| n_dead_tup | Numeric | Number of dead tuples |
| page_dirty_rate | numeric(5,2) | Dirty page rate (%) of a table |

Example

Query partition tables whose dirty page rate is greater than 30%.

```

SELECT * FROM PGXC_GET_STAT_ALL_PARTITIONS WHERE dirty_page_rate>30;
relid | partid | schemaname | relname | partname | n_tup_ins | n_tup_upd | n_tup_del | n_live_tup |
n_dead_tup | dirty_page_rate

```

| | | | | | | | | | |
|--------|-----------|-----------------|-----------------|--------------------|----------------|----|---|---|---|
| 58320 | 58626 | schema_subquery | store_hash_par | p1 | 2 | 0 | 2 | 0 | 2 |
| 100.00 | 58430 | 58706 | schema_subquery | store_hash_par_mor | p4 | 1 | 1 | 1 | 0 |
| 2 | 100.00 | 58320 | 58644 | schema_subquery | store_hash_par | p1 | 3 | 0 | 3 |
| 100.00 | 58430 | 58770 | schema_subquery | store_hash_par_mor | p4 | 1 | 1 | 1 | 0 |
| 2 | 100.00 | 58320 | 58643 | schema_subquery | store_hash_par | p1 | 2 | 0 | 2 |
| 100.00 | 58320 | 58625 | schema_subquery | store_hash_par | p1 | 2 | 0 | 2 | 0 |
| 100.00 | 58320 | 58579 | schema_subquery | store_hash_par | p1 | 2 | 0 | 2 | 0 |
| 100.00 | 58320 | 58619 | schema_subquery | store_hash_par | p1 | 3 | 0 | 3 | 0 |
| 100.00 | 58320 | 58627 | schema_subquery | store_hash_par | p1 | 4 | 0 | 4 | 0 |
| 100.00 | 58320 | 58657 | schema_subquery | store_hash_par | p1 | 3 | 0 | 3 | 0 |
| 100.00 | (10 rows) | | | | | | | | |

18.3.195 PGXC_GET_TABLE_SKEWNESS

PGXC_GET_TABLE_SKEWNESS displays the data skew on tables in the current database. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

You are advised to use **PGXC_GET_TABLE_SKEWNESS** to query data skew when there are less than 10,000 tables in a database.

Table 18-269 PGXC_GET_TABLE_SKEWNESS columns

| Column | Type | Description |
|------------|------------------|--|
| schemaname | Name | Schema name of a table |
| tablename | Name | Name of a table |
| totalsize | Numeric | Total size of a table, in bytes |
| avgsz | numeric(1000, 0) | Average table size (total table size divided by the number of DNs), which is the ideal size of tables distributed on each DN |
| maxratio | numeric(10,3) | Ratio of the maximum table size on a single DN to the value of avgsz . |
| minratio | numeric(10,3) | Ratio of the minimum table size on a single DN to avgsz |
| skewsize | Bigint | Table skew rate (the maximum table size on a single DN minus the minimum table size on a single DN) |
| skewratio | numeric(10,3) | Table skew rate (skewsize/avgsz) |

| Column | Type | Description |
|------------|------------------|--|
| skewstddev | numeric(1000, 0) | Standard deviation of table distribution (For two tables of the same size, a larger deviation indicates a more severe skew.) |

18.3.196 PGXC_GLOBAL_TEMP_ATTACHED_PIDS

This view displays information about sessions of resources occupied by global temporary tables on CNs. This view is supported only by clusters of 8.2.1.220 and later versions.

Table 18-270 PG_GLOBAL_TEMP_ATTACHED_PIDS columns

| Column | Type | Description |
|------------|--------|------------------|
| nodename | Name | Node name |
| schemaname | Name | Schema name |
| tablename | Name | Table name |
| pid | Bigint | PID of a session |

18.3.197 PGXC_GTM_SNAPSHOT_STATUS

PGXC_GTM_SNAPSHOT_STATUS displays transaction information on the current GTM.

Table 18-271 PGXC_GTM_SNAPSHOT_STATUS columns

| Column | Type | Description |
|--------------|---------|--|
| xmin | Xid | Minimum ID of the running transactions |
| xmax | Xid | ID of the transaction next to the executed transaction with the maximum ID |
| csn | Integer | Sequence number of the transaction to be committed |
| oldestxmin | Xid | Minimum ID of the executed transactions |
| xcnt | Integer | Number of the running transactions |
| running_xids | Text | IDs of the running transactions |

18.3.198 PGXC_INSTANCE_TIME

PGXC_INSTANCE_TIME displays the running time of processes on each node in the cluster and the time consumed in each execution phase. Except the **node_name** column, the other columns are the same as those in the [PV_INSTANCE_TIME](#) view. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 18-272 PGXC_INSTANCE_TIME columns

| Column | Type | Description |
|-----------|---------|--------------------------|
| node_name | Text | Node name |
| stat_id | Integer | Type ID |
| stat_name | Text | Name of the runtime type |
| value | Bigint | Runtime value |

18.3.199 PGXC_LOCKWAIT_DETAIL

PGXC_LOCKWAIT_DETAIL displays detailed information about the lock wait hierarchy on each node in a cluster. If a node has multiple lock wait levels, the entire lock waiting hierarchy is displayed in sequence.

This view is supported only by clusters of version 8.1.3.200 or later.

Table 18-273 PGXC_LOCKWAIT_DETAIL columns

| Column | Type | Description |
|---------------------|----------|---|
| level | Integer | Level in the lock wait hierarchy. The value starts with 1 and increases by 1 when there is a wait relationship. |
| node_name | Name | Node name, corresponding to the node_name column in the pgxc_node table. |
| lock_wait_hierarchy | Text | Lock wait hierarchy, in the format of <i>node-name.process-ID->waiting-process-ID->waiting-process-ID->...</i> |
| lock_type | Text | Type of the locked object |
| database | OID | OID of the database where the locked object is. |
| relation | OID | OID of the relationship of the locked object. |
| page | Integer | Page index in a relationship |
| tuple | Smallint | Row number of a page. |
| virtual_xid | Text | Virtual ID of a transaction. |

| Column | Type | Description |
|---------------------|--------------------------|--|
| transaction_id | Xid | Transaction ID. |
| class_id | OID | OID of the system catalog that contains the object. |
| obj_id | OID | OID of the object within its system catalog. |
| obj_subid | Smallint | Column number of a table |
| virtual_transaction | Text | Virtual ID of the transaction holding or waiting for the lock. |
| pid | Bigint | ID of the thread holding or awaiting this lock |
| mode | Text | Lock level |
| granted | Boolean | Indicates whether a lock is held. |
| fastpath | Boolean | Indicates whether to obtain a lock using FASTPATH. |
| wait_for_pid | Bigint | ID of the thread where a lock conflict occurs. |
| conflict_mode | Text | Level of the conflicted lock held by the thread where it is |
| query_id | Bigint | ID of a query statement. |
| query | Text | Query statement |
| application_name | Text | Name of the application connected to the backend |
| backend_start | Timestamp with time zone | Startup time of the backend process, that is, the time when the client connects to the server |
| xact_start | Timestamp with time zone | Start time of the current transaction |
| query_start | Timestamp with time zone | Start time of the active query |
| state | Text | Overall state of the backend |
| waittime | Timestamp with time zone | Timestamp when the lock wait starts.
This column is available only in clusters of version 9.1.0.200 or later. |
| holdtime | Timestamp with time zone | Timestamp when the lock starts to be obtained.
This column is available only in clusters of version 9.1.0.200 or later. |

Example

Step 1 Connect to the DN, start a transaction, and run the following command:

```
begin;select * from t1;
```

Step 2 Connect to the CN in another window and truncate table **t1**.

```
truncate t1;
```

In this case, truncation is blocked.

Step 3 Open another window to connect to the CN and run the **select * from pgxc_lockwait_detail;** command.

```
SELECT * FROM PGXC_LOCKWAIT_DETAIL;
level | node_name | lock_wait_hierarchy | lock_type | database | relation | page | tuple |
virtual_xid | transaction_id | class_id | obj_id | obj_subid | virtual_transaction | p
id | mode | granted | fastpath | wait_for_pid | conflict_mode | query_id
| query | application_name | backend_start |
xact_start | query_start | state
-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
1 | datanode1 | datanode1:140378619314976 | relation | 16049 | 2147484411 | |
| 673638 | | | 19/297 | 1403786
19314976 | AccessExclusiveLock | f | f | 140378619263840 | AccessShareLock | 73183493945504391
| TRUNCATE t1 | coordinator1 | 2023-03-13 12:13:52.530602+08 | 2
023-03-13 14:52:16.1456+08 | 2023-03-13 14:52:16.148693+08 | active
2 | datanode1 | datanode1:140378619314976 -> 140378619263840 | relation | 16049 | 2147484411 |
| 19263840 | AccessShareLock | t | f | 23/16067 | 1403786
| gsq | 2023-03-13 14:19:26.325602+08 | 2
023-03-13 14:52:12.042741+08 | 2023-03-13 14:52:12.042741+08 | idle in transaction
(2 rows)
```

----End

18.3.200 PGXC_INSTR_UNIQUE_SQL

PGXC_INSTR_UNIQUE_SQL displays the complete Unique SQL statistics of all CN nodes in the cluster.

Only the system administrator can access this view. The columns in this view are the same as those in the **GS_INSTR_UNIQUE_SQL** view. For details about columns in the view, see [Table 18-274](#).

Table 18-274 GS_INSTR_UNIQUE_SQL columns

| Column | Type | Description |
|-----------|---------|---|
| node_name | Name | Name of the CN that receives SQL statements |
| node_id | Integer | Node ID, which is the same as the value of node_id in the pgxc_node table |

| Column | Type | Description |
|-------------------|--------|---|
| user_name | Name | Username |
| user_id | OID | User ID |
| unique_sql_id | Bigint | Normalized Unique SQL ID |
| query | Text | Normalized SQL text |
| n_calls | Bigint | Number of successful execution times |
| min_elapse_time | Bigint | Minimum running time of the SQL statement in the database (unit: μ s) |
| max_elapse_time | Bigint | Maximum running time of SQL statements in the database (unit: μ s) |
| total_elapse_time | Bigint | Total running time of SQL statements in the database (unit: μ s) |
| n_returned_rows | Bigint | Row activity - Number of rows in the result set returned by the SELECT statement |
| n_tuples_fetched | Bigint | Row activity - Randomly scan rows (column-store tables/foreign tables are not counted.) |
| n_tuples_returned | Bigint | Row activity - Sequential scan rows (Column-store tables/foreign tables are not counted.) |
| n_tuples_inserted | Bigint | Row activity - Inserted rows |
| n_tuples_updated | Bigint | Row activity - Updated rows |
| n_tuples_deleted | Bigint | Row activity - Deleted rows |
| n_blocks_fetched | Bigint | Block access times of the buffer, that is, physical read/I/O |
| n_blocks_hit | Bigint | Block hits of the buffer, that is, logical read/cache |

| Column | Type | Description |
|---------------------|--------|--|
| n_soft_parse | Bigint | Number of soft parsing times (cache plan) |
| n_hard_parse | Bigint | Number of hard parsing times (generation plan) |
| db_time | Bigint | Valid DB execution time, including the waiting time and network sending time. If multiple threads are involved in query execution, the value of DB_TIME is the sum of DB_TIME of multiple threads (unit: μ s). |
| cpu_time | Bigint | CPU execution time, excluding the sleep time (unit: μ s) |
| execution_time | Bigint | SQL execution time in the query executor, DDL statements, and statements (such as Copy statements) that are not executed by the executor are not counted (unit: μ s). |
| parse_time | Bigint | SQL parsing time (unit: μ s) |
| plan_time | Bigint | SQL generation plan time (unit: μ s) |
| rewrite_time | Bigint | SQL rewriting time (unit: μ s) |
| pl_execution_time | Bigint | Execution time of the plpgsql procedural language function (unit: μ s) |
| pl_compilation_time | Bigint | Compilation time of the plpgsql procedural language function (unit: μ s) |

| Column | Type | Description |
|---------------|--------------------------|---|
| net_send_time | Bigint | Network time, including the time spent by the CN in sending data to the client and the time spent by the DN in sending data to the CN (unit: μ s) |
| data_io_time | Bigint | File I/O time (unit: μ s) |
| first_time | Timestamp with time zone | Time of the first SQL statement execution |
| last_time | Timestamp with time zone | Time of the last SQL statement execution |

18.3.201 PGXC_LOCK_CONFLICTS

PGXC_LOCK_CONFLICTS displays information about conflicting locks in the cluster.

When a lock is waiting for another lock or another lock is waiting for this one, a lock conflict occurs.

Currently, **PGXC_LOCK_CONFLICTS** collects only information about locks whose **locktype** is **relation**, **partition**, **page**, **tuple**, or **transactionid**.

Table 18-275 PGXC_LOCK_CONFLICTS columns

| Column | Type | Description |
|----------|------|--|
| locktype | Text | Type of the locked object |
| nodename | Name | Name of the node where the locked object resides |
| dbname | Name | Name of the database where the locked object resides. The value is NULL if the locked object is a transaction. |
| nspname | Name | Name of the namespace of the locked object |
| relname | Name | Name of the relation targeted by the lock. The value is NULL if the object is not a relation or part of a relation. |
| partname | Name | Name of the partition targeted by the lock. The value is NULL if the locked object is not a partition. |

| Column | Type | Description |
|---------------|--------------------------|--|
| page | Integer | Number of the page targeted by the lock.
The value is NULL if the locked object is neither a page nor a tuple. |
| tuple | Smallint | Number of the tuple targeted by the lock.
The value is NULL if the locked object is not a tuple. |
| transactionid | Xid | ID of the transaction targeted by the lock.
The value is NULL if the locked object is not a transaction. |
| username | Name | Name of the user who applies for the lock |
| gxid | Xid | ID of the transaction that applies for the lock |
| xactstart | Timestamp with time zone | Start time of the transaction that applies for the lock |
| queryid | Bigint | Latest query ID of the thread that applies for the lock |
| query | Text | Latest query statement of the thread that applies for the lock |
| pid | Bigint | ID of the thread that applies for the lock |
| mode | Text | Lock mode |
| granted | Boolean | <ul style="list-style-type: none">• TRUE if the lock has been held• FALSE if the lock is still waiting for another lock |

18.3.202 PGXC_LWLOCKS

PGXC_LWLOCK offers details on lightweight locks that are currently held or being waited for by all instances in the cluster. This view is supported only by 9.1.0.200 and later cluster versions.

Table 18-276 PGXC_LWLOCKS columns

| Column | Type | Description |
|----------|---------|--|
| nodename | Name | Name of the node where the locked object resides |
| pid | Bigint | ID of the backend thread |
| query_id | Bigint | ID of a query |
| lwtid | Integer | Lightweight thread ID of the backend thread |

| Column | Type | Description |
|--------------|---------|--|
| reqlockid | Integer | ID of the lightweight lock that is being requested by the current thread |
| reqlock | Text | Name of the lightweight lock corresponding to reqlockid |
| heldlocknums | Integer | Number of lightweight locks obtained by the current thread |
| heldlockid | Integer | Lightweight lock ID obtained by the current thread |
| heldlock | Text | Name of the lightweight lock corresponding to heldlockid |
| heldlockmode | Text | Lightweight lock mode corresponding to heldlockid |

Example

Use the **PGXC_LWLOCKS** view to get details on lightweight locks that are currently held or being waited for by all instances in the cluster.

```
SELECT * FROM pgxc_lwlocks;
nodename | pid | query_id | lwtid | reqlockid | reqlock | heldlocknums | heldlockid |
heldlock | heldlockmode
-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+
datanode1 | 139810224193360 | 78250043525924188 | 54844 | | | 1 | 76390 |
BUFFER_POOL_LWLOCK | Shared
datanode1 | 139810224198200 | 78250043525924886 | 54922 | | | 1 | 957438 |
PGPROC_LWLOCK | Exclusive
datanode2 | 140262654050288 | 0 | 54832 | | | 1 | 7 |
WALWriteLock | Exclusive
datanode2 | 140262654052488 | 78250043525923195 | 54847 | | | 1 | 15862 |
BUFFER_POOL_LWLOCK | Shared
(4 rows)
```

18.3.203 PGXC_MEMORY_DEBUG_INFO

PGXC_MEMORY_DEBUG_INFO displays memory error information for each node in the current cluster when executing jobs, making it easy to locate memory error issues. When an error message "memory is temporarily unavailable" is prompted during statement execution, this view can be used to query memory error information for all nodes, which is the same as the memory error information displayed in the log. This view is supported only by clusters of version 8.3.0 or later.

NOTICE

This view only displays the most recent cluster information for errors, and repeated error information will be overwritten. If the same query requests memory multiple times and errors occur, the information will not be updated.

Table 18-277 PGXC_MEMORY_DEBUG_INFO columns

| Column | Type | Description |
|-------------|--------|--|
| node_name | Text | Instance name, including CNs and DN. |
| query_id | Bigint | ID of the query that is currently requesting memory. |
| memory_info | Text | Current instance's memory usage, including: <ul style="list-style-type: none">• process_used_memory: memory size used by the DWS process.• max_dynamic_memory: maximum dynamic memory.• dynamic_used_memory: used dynamic memory.• dynamic_peak_memory: dynamic peak value of memory.• dynamic_used_shrctx: maximum dynamic shared memory context.• dynamic_peak_shrctx: dynamic peak value of shared memory context.• shared_used_memory: used shared memory.• cstore_used_memory: memory size used for column store.• comm_used_memory: memory size used by the communication library.• comm_peak_memory: peak value of memory used by the communication library.• other_used_memory: memory size used by other components.• topsql_used_memory: memory size used by topsql.• large_storage_memory: memory size used for column-store compression and decompression.• os_totalmem: total memory size of the operating system.• os_freemem: remaining memory size of the operating system. |
| summary | Text | The total estimated memory consumption and actual memory consumption of jobs on the instance. |

| Column | Type | Description |
|-----------------|--------------------------|--|
| abnormal_query | Text | Thread ID and query ID with abnormal memory usage, including two cases:
1. Session with the maximum current memory usage.
2. Session with the largest difference between estimated memory and actual memory usage. |
| abnormal_memory | Text | Memory block with the highest usage, including the maximum shared memory context usage and the maximum common memory context usage. |
| top_thread | Text | Information on the top three threads with the highest memory usage:
context name : memory block currently in use.
contextlevel : context level.
sessType : type of the top-level context node.
totalsize[274,13,260]MB : total memory, released memory, and used memory size of the current memory context, in MB. |
| create_time | Timestamp with time zone | Time when the memory shortage error occurred. |

18.3.204 PGXC_NODE_ENV

PGXC_NODE_ENV displays the environmental variables information about all nodes in a cluster.

Table 18-278 PGXC_NODE_ENV columns

| Column | Type | Description |
|---------------|---------|---|
| node_name | Text | Names of all nodes in the cluster. |
| host | Text | Host names of all nodes in the cluster. |
| process | Integer | Process IDs of all nodes in the cluster. |
| port | Integer | Port numbers of all nodes in the cluster. |
| installpath | Text | Installation directory of all nodes in the cluster. |
| datapath | Text | Data directories of all nodes in the cluster. |
| log_directory | Text | Log directories of all nodes in the cluster. |

18.3.205 PGXC_NODE_STAT_RESET_TIME

PGXC_NODE_STAT_RESET_TIME displays the time when statistics of each node in the cluster are reset. All columns except **node_name** are the same as those in the [GS_NODE_STAT_RESET_TIME](#) view. This view is accessible only to users with system administrators rights.

Table 18-279 PGXC_NODE_STAT_RESET_TIME columns

| Column | Type | Description |
|------------|-----------|---|
| node_name | Text | Node name |
| reset_time | Timestamp | Time when statistics on each node are reset |

18.3.206 PGXC_OBS_IO_SCHEDULER_STATS

PGXC_OBS_IO_SCHEDULER_STATS displays the latest real-time statistics about read/write requests of the OBS I/O Scheduler. This system view is supported only by clusters of version 9.1.0 or later.

Table 18-280 PGXC_OBS_IO_SCHEDULER_STATS columns

| Column | Type | Description |
|---------------------|------|--|
| node_name | Text | Node name. |
| io_type | Char | Type of I/O operation, including: <ul style="list-style-type: none">• r: read.• w: write.• s: file operation. |
| current_bps | INT8 | Current bandwidth rate, in KB/s. |
| best_bps | INT8 | Best bandwidth rate achieved recently, in KB/s. |
| waiting_request_num | Int | Number of queued requests currently waiting. |
| mean_request_size | INT8 | Average length of requests processed recently, in KB. |
| total_token_num | Int | Total number of I/O tokens. |
| available_token_num | Int | Number of available I/O tokens. |
| total_worker_num | Int | Total number of working threads. |
| idle_worker_num | Int | Number of idle working threads. |

Example

Step 1 Query statistics about read requests of OBS I/O Scheduler on each node:

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;
```

| node_name | io_type | current_bps | best_bps | waiting_request_num | mean_request_size | total_token_num | available_token_num | total_worker_num | idle_worker_num |
|--------------|---------|-------------|----------|---------------------|-------------------|-----------------|---------------------|------------------|-----------------|
| dn_6001_6002 | r | 26990 | 26990 | 0 | 215 | 18 | 16 | | |
| dn_6003_6004 | r | 21475 | 21475 | 10 | 190 | 30 | 30 | | |
| dn_6005_6006 | r | 12384 | 12384 | 36 | 133 | 30 | 27 | | |

According to the result, this is a snapshot of the statistics at a certain time point when the current I/O scheduler reads I/Os. At this time, the bandwidth is increasing, and **current_bps** is equal to **best_bps**. Take **dn_6003_6004** as an example. You can see that there are queuing requests on the current DN. The value of **total_token_num** is the same as that of **available_token_num**, indicating that the I/O scheduler has not started to process these requests when the view is queried.

Step 2 Wait for a while and initiate the query again.

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;
```

| node_name | io_type | current_bps | best_bps | waiting_request_num | mean_request_size | total_token_num | available_token_num | total_worker_num | idle_worker_num |
|--------------|---------|-------------|----------|---------------------|-------------------|-----------------|---------------------|------------------|-----------------|
| dn_6001_6002 | r | 13228 | 26990 | 0 | 609 | 18 | 18 | | |
| dn_6003_6004 | r | 15717 | 21475 | 0 | 622 | 30 | 30 | | |
| dn_6005_6006 | r | 18041 | 21767 | 0 | 609 | 30 | 30 | | |

When the queue is empty and the value of **available_token_num** is equal to that of **total_token_num**, it indicates that the I/O scheduler has finished processing all requests and there are no new requests in line. The **current_bps** value is not 0 because it represents the average bandwidth (in bit/s) over a three-second period. Therefore, the displayed value reflects the data from three seconds ago.

Step 3 After a short period of time, the query result is as follows. The value of **current_bps** changes to 0.

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;
```

| node_name | io_type | current_bps | best_bps | waiting_request_num | mean_request_size | total_token_num | available_token_num | total_worker_num | idle_worker_num |
|--------------|---------|-------------|----------|---------------------|-------------------|-----------------|---------------------|------------------|-----------------|
| dn_6001_6002 | r | 0 | 26990 | 0 | 609 | 18 | 18 | | |
| dn_6003_6004 | r | 0 | 21475 | 0 | 622 | 30 | 30 | | |
| dn_6005_6006 | r | 0 | 21767 | 0 | 609 | 30 | 30 | | |

----End

18.3.207 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS

PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS provides statistics on the number of requests and flow control information for different types of OBS I/O Scheduler requests, including read, write, and file operations. This system view is supported only by clusters of version 9.1.0 or later.

The first query result shows the statistics from the cluster startup to the query time, with detailed columns listed in the table below.

Table 18-281 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS columns

| Column | Type | Description |
|--------------------------|--------------|---|
| node_name | Name | Name of a CN or DN, for example, dn_6001_6002 . |
| io_type | Char | Type of I/O operation, including: <ul style="list-style-type: none">• R: read• W: write• S: file operation |
| recent_throttled_req_num | Int | Number of times flow control was applied between two query views. |
| total_throttled_req_num | Int | Total number of times flow control was applied. |
| last_throttled_dur(s) | INT8 | Time interval since the last occurrence of flow control. |
| waiting_req_num | Int | Number of queued requests currently waiting. |
| mean_tps | numeric(7,2) | Average TPS (transactions per second) between two query views. |
| mean_req_size(KB) | INT8 | Average length of requests between two query views, in KB. |
| mean_req_latency(ms) | INT8 | Average latency of requests between two query views, in ms. |
| max_req_latency(ms) | INT8 | Maximum latency of requests before two query views, in ms. |

| Column | Type | Description |
|----------------|------|---|
| mean_bps(KB/s) | INT8 | Average read or write speed between two query views, in KB/s. |
| duration(s) | Int | Time interval between two query views, in seconds. |

Example

Run the **SELECT * FROM pgxc_obs_io_scheduler_periodic_stats** statement to query the view content. The following is an example of the query result:

```
SELECT * FROM pgxc_obs_io_scheduler_periodic_stats;
```

| node_name | io_type | recent_throttled_req_num | total_throttled_req_num | last_throttled_dur(s) | waiting_req_num | mean_tps | mean_req_size(KB) | mean_req_latency(ms) | max_req_latency(ms) | mean_bps(KB/s) | duration(s) |
|-----------|---------|--------------------------|-------------------------|-----------------------|-----------------|----------|-------------------|----------------------|---------------------|----------------|-------------|
|-----------|---------|--------------------------|-------------------------|-----------------------|-----------------|----------|-------------------|----------------------|---------------------|----------------|-------------|

| dn_6001_6002 S | | dn_6001_6002 R | | dn_6001_6002 W | | cn_5001 S | | cn_5001 R | | cn_5001 W | |
|------------------|------|------------------|------|------------------|------|-------------|-----|-------------|------|-------------|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 207 | 0 | 0 | 0 | 288 |
| 155 | 155 | 155 | 155 | 155 | 155 | 155 | 519 | 155 | 155 | 155 | 288 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | .03 | 0.00 | 0.00 | .01 | 0.00 |

To display **0** before the decimal point in the value of **mean_tps**, set the **display_leading_zero** option in the **behavior_compat_options** parameter.

Run the **select * from pgxc_obs_io_scheduler_periodic_stats** statement. The following information is displayed:

```
SELECT * FROM pgxc_obs_io_scheduler_periodic_stats;
```

| node_name | io_type | recent_throttled_req_num | total_throttled_req_num | last_throttled_dur(s) | waiting_req_num | mean_tps | mean_req_size(KB) | mean_req_latency(ms) | max_req_latency(ms) | mean_bps(KB/s) | duration(s) |
|-----------|---------|--------------------------|-------------------------|-----------------------|-----------------|----------|-------------------|----------------------|---------------------|----------------|-------------|
| node1 | read | 100 | 100 | 0.001 | 0 | 1000 | 1024 | 100 | 1000 | 1000 | 1000 |
| node2 | write | 50 | 50 | 0.001 | 0 | 500 | 512 | 50 | 500 | 500 | 500 |
| node3 | read | 200 | 200 | 0.001 | 0 | 2000 | 2048 | 200 | 2000 | 2000 | 2000 |
| node4 | write | 75 | 75 | 0.001 | 0 | 750 | 768 | 75 | 750 | 750 | 750 |
| node5 | read | 150 | 150 | 0.001 | 0 | 1500 | 1536 | 150 | 1500 | 1500 | 1500 |
| node6 | write | 60 | 60 | 0.001 | 0 | 600 | 614 | 60 | 600 | 600 | 600 |
| node7 | read | 180 | 180 | 0.001 | 0 | 1800 | 1856 | 180 | 1800 | 1800 | 1800 |
| node8 | write | 40 | 40 | 0.001 | 0 | 400 | 409 | 40 | 400 | 400 | 400 |
| node9 | read | 120 | 120 | 0.001 | 0 | 1200 | 1228 | 120 | 1200 | 1200 | 1200 |
| node10 | write | 90 | 90 | 0.001 | 0 | 900 | 921 | 90 | 900 | 900 | 900 |

| dn_6001_6002 | | S | | R | | W | | 0 | | 0.36 | |
|--------------|---|-----|---|-----|---|-----|---|---|---|------|--|
| dn_6001_6002 | S | 132 | 0 | 326 | 0 | 177 | 0 | 0 | 0 | 0.36 | |
| dn_6001_6002 | R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0.00 | |
| dn_6001_6002 | W | 0 | 0 | 0 | 0 | 177 | 0 | 0 | 0 | 0.00 | |
| cn_5001 | S | 0 | 0 | 0 | 0 | 177 | 0 | 0 | 0 | 0.00 | |
| cn_5001 | R | 0 | 0 | 0 | 0 | 177 | 0 | 0 | 0 | 0.00 | |
| cn_5001 | W | 0 | 0 | 0 | 0 | 177 | 0 | 0 | 0 | 0.00 | |

18.3.208 PGXC_OS_RUN_INFO

PGXC_OS_RUN_INFO displays the OS running status of each node in the cluster. All columns except **node_name** are the same as those in the [PV_OS_RUN_INFO](#) view. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 18-282 PGXC_OS_RUN_INFO field columns

| Column | Type | Description |
|------------|---------|--|
| node_name | Text | Node name |
| id | Integer | ID |
| Name | Text | Name of the OS running status |
| value | Numeric | Value of the OS status |
| comments | Text | Remarks of the OS status |
| cumulative | boolean | Whether the value of the OS status is cumulative |

18.3.209 PGXC_OS_THREADS

PGXC_OS_THREADS displays thread status information under all normal nodes in the current cluster.

Table 18-283 PGXC_OS_THREADS columns

| Column | Type | Description |
|---------------|--------------------------|---|
| node_name | Text | Names of all normal nodes currently in the cluster. |
| pid | Bigint | Thread IDs currently running in the processes of all normal nodes in the cluster. |
| lwpid | Integer | Lightweight thread IDs corresponding to the PIDs. |
| thread_name | Text | Thread names corresponding to the PIDs. |
| creation_time | Timestamp with time zone | Creation time of the threads corresponding to the PIDs. |

18.3.210 PGXC_POOLER_STATUS

PGXC_POOLER_STATUS displays the pooler cache connection status of each CN in the current cluster. This view can be queried only on CNs to display the connection

cache information of the pooler module on all CNs. The **PGXC_POOLER_STATUS** view is supported only by clusters of version 8.3.0 or later.

Table 18-284 PGXC_POOLER_STATUS columns

| Column | Type | Description |
|----------------|---------|---|
| coorname | Text | Name of the CN node. |
| database | Text | Database name. |
| user_name | Text | Username. |
| tid | Bigint | ID of the thread used for the connection to the CN. |
| node_oid | Bigint | OID of the node connected to. |
| node_name | Name | Name of the node connected to. |
| in_use | boolean | Whether the connection is currently in use. The options are: <ul style="list-style-type: none">• t (true): The connection is in use.• f (false): The connection is not in use. |
| fdsock | Bigint | Peer socket. |
| remote_pid | Bigint | Peer thread ID. |
| session_params | Text | GUC session parameters issued by this connection. |

18.3.211 PGXC_PREPARED_XACTS

PGXC_PREPARED_XACTS displays the two-phase transactions in the **prepared** phase.

Table 18-285 PGXC_PREPARED_XACTS columns

| Column | Type | Description |
|--------------------|------|--|
| pgxc_prepared_xact | Text | Two-phase transactions in the prepared phase. |

18.3.212 PGXC_REDO_STAT

PGXC_REDO_STAT displays statistics on redoing Xlogs of each node in the cluster. All columns except **node_name** are the same as those in the [PV_REDO_STAT](#) view. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 18-286 PGXC_REDO_STAT columns

| Column | Type | Description |
|-----------|--------|-----------------------------------|
| node_name | Text | Node name |
| phywrts | Bigint | Number of physical writes |
| phyblkwrt | Bigint | Number of physical blocks written |
| writetim | Bigint | Time taken for physical writes |
| avgiotim | Bigint | Average time taken per write |
| lstiotim | Bigint | Time taken for the last write |
| miniotim | Bigint | Minimum time taken for a write |
| maxiowtm | Bigint | Maximum time taken for a write |

18.3.213 PGXC_REL_IOSTAT

PGXC_REL_IOSTAT displays disk read/write statistics on each node in the cluster. This view is accessible only to users with system administrators rights.

Table 18-287 PGXC_REL_IOSTAT columns

| Column | Type | Description |
|-----------|--------|-------------------------|
| node_name | Text | Node name |
| phyrds | Bigint | Number of disk reads |
| phywrts | Bigint | Number of disk writes |
| phyblkrd | Bigint | Number of read pages |
| phyblkwrt | Bigint | Number of written pages |

18.3.214 PGXC_REPLICATION_SLOTS

PGXC_REPLICATION_SLOTS displays the replication information of DNs in the cluster. All columns except **node_name** are the same as those in the [PG_REPLICATION_SLOTS](#) view. This view is accessible only to users with system administrators rights.

Table 18-288 PGXC_REPLICATION_SLOTS columns

| Column | Type | Description |
|-----------|------|----------------------------|
| node_name | Text | Node name |
| slot_name | Text | Name of a replication node |

| Column | Type | Description |
|---------------|---------|--|
| plugin | Name | Name of the output plug-in of the logical replication slot |
| slot_type | Text | Type of a replication node |
| datoid | OID | OID of the database on the replication node |
| database | Name | Name of the database on the replication node |
| active | boolean | Whether the replication node is active |
| xmin | Xid | Transaction ID of the replication node |
| catalog_xmin | Text | ID of the earliest-decoded transaction corresponding to the logical replication slot |
| restart_lsn | Text | Xlog file information on the replication node |
| dummy_standby | boolean | Whether the replication node is the dummy standby node |

18.3.215 PGXC_RESPOOL_RUNTIME_INFO

PGXC_RESPOOL_RUNTIME_INFO displays the running information about all resource pool jobs on all CNs.

Table 18-289 PGXC_RESPOOL_RUNTIME_INFO columns

| Column | Type | Description |
|-----------|------|---|
| nodename | Name | CN name. |
| nodegroup | Name | Name of the logical cluster the resource pool belongs to. The default cluster is installation . |
| rpname | Name | Resource pool name. |
| ref_count | Int | Number of jobs that reference the resource pool. This count includes both controlled and uncontrolled jobs. |
| fast_run | Int | Number of jobs currently running in the resource pool's fast lane. |
| fast_wait | Int | Number of jobs currently queued in the resource pool's fast lane. |
| slow_run | Int | Number of jobs currently running in the resource pool's slow lane. |
| slow_wait | Int | Number of jobs currently queued in the resource pool's slow lane. |

18.3.216 PGXC_RESPOOL_RESOURCE_INFO

PGXC_RESPOOL_RESOURCE_INFO displays the real-time monitoring information about the resource pools on all instances.

NOTE

- On a DN, it only displays the monitoring information of the logical cluster that the DN belongs to.
- Cluster 8.2.0 and later versions provide the negative memory feedback mechanism. The CCN can decrease the estimated memory usage of statements based on their actual memory usage on DNs, improving resource utilization by reducing overestimation. However, the estimated memory usage on CNs remains unchanged. If the CCN allows more jobs to run, the total estimated memory usage in the resource pool monitoring view may exceed the memory upper limit of the resource pool.
- Only the operators occupying large memory are under statement memory control. The memory, thread initialization costs, and expression costs of the operators with small memory usage are not controlled. So the value of **used_mem** of the resource pool may exceed the value of **mem_limit** to a limited extent.

Table 18-290 PGXC_RESPOOL_RESOURCE_INFO columns

| Column | Type | Description |
|------------|------|--|
| nodename | Name | Instance name, including CNs and DNs. |
| nodegroup | Name | Name of the logical cluster of the resource pool. The default value is installation . |
| rpname | Name | Resource pool name. |
| cgroup | Name | Name of the Cgroup associated with the resource pool. |
| ref_count | Int | Number of jobs referenced by the resource pool. The number is counted regardless of whether the jobs are controlled by the resource pool. This parameter is valid only on CNs. |
| fast_run | Int | Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs. |
| fast_wait | Int | Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs. |
| fast_limit | Int | Limit on the number of concurrent jobs in the fast lane in a resource pool. This parameter is valid only on CNs. |
| slow_run | Int | Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs. |

| Column | Type | Description |
|-----------------|--------|---|
| slow_wait | Int | Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs. |
| slow_limit | Int | Limit on the number of concurrent jobs in the slow lane in a resource pool. This parameter is valid only on CNs. |
| used_cpu | Double | Average number of CPUs used by the resource pool in a 5s monitoring period. The value is accurate to two decimal places. <ul style="list-style-type: none">On a DN, it indicates the number of CPUs used by the resource pool on the current DN.On a CN, it indicates the total CPU usage of resource pools on all DNs. |
| cpu_limit | Int | It indicates the upper limit of available CPUs for resource pools. If the CPU share is limited, this parameter indicates the available CPUs for DWS. If the CPU limit is specified, this parameter indicates the available CPUs for associated Cgroups. <ul style="list-style-type: none">On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN.On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs. |
| used_mem | Int | Memory size used by the resource pool (unit: MB) <ul style="list-style-type: none">On a DN, it indicates the memory usage of the resource pool on the current DN.On a CN, it indicates the total memory usage of resource pools on all DNs. |
| estimate_memory | Int | Estimated memory used by the jobs running in the resource pools on the current CN. This parameter is valid only on CNs. |
| mem_limit | Int | Upper limit of available memory for the resource pool (unit: MB). <ul style="list-style-type: none">On a DN, it indicates the upper limit of available memory for the resource pool on the current DN.On a CN, it indicates the total upper limit of available memory for resource pools on all DNs. |

| Column | Type | Description |
|--------------|--------|--|
| read_kbytes | Bigint | Number of logical read bytes in the resource pool within a 5s monitoring period (unit: KB). <ul style="list-style-type: none">On a DN, it indicates the number of logical read bytes in the resource pool on the current DN.On a CN, it indicates the total logical read bytes of resource pools on all DNs. |
| write_kbytes | Bigint | Number of logical write bytes in the resource pool within a 5s monitoring period (unit: KB). <ul style="list-style-type: none">On a DN, it indicates the number of logical write bytes in the resource pool on the current DN.On a CN, it indicates the total logical write bytes of resource pools on all DNs. |
| read_counts | Bigint | Number of logical reads in the resource pool within a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the number of logical reads in the resource pool on the current DN.On a CN, it indicates the total number of logical reads in resource pools on all DNs. |
| write_counts | Bigint | Number of logical writes in the resource pool within a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the number of logical writes in the resource pool on the current DN.On a CN, it indicates the total number of logical writes in resource pools on all DNs. |
| read_speed | Double | Average logical read rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the logical read rate of the resource pool on the current DN.On a CN, it indicates the overall logical read rate of resource pools on all DNs. |
| write_speed | Double | Average logical write rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the logical write rate of the resource pool on the current DN.On a CN, it indicates the overall logical write rate of resource pools on all DNs. |

| Column | Type | Description |
|------------|--------|---|
| send_speed | Double | Average network sending rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the network sending rate of the resource pool on the current DN.On a CN, it indicates the sum of the network sending rates of the resource pool on all DNs. |
| recv_speed | Double | Average network sending rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the network sending rate of the resource pool on the current DN.On a CN, it indicates the sum of the network sending rates of the resource pool on all DNs. |

18.3.217 PGXC_RESPOOL_RESOURCE_HISTORY

PGXC_RESPOOL_RESOURCE_HISTORY is used to query historical monitoring information about resource pools on all instances.

Table 18-291 PGXC_RESPOOL_RESOURCE_HISTORY columns

| Column | Type | Description |
|-----------|-----------|--|
| nodename | Name | Instance name, including CNs and DNs |
| Timestamp | Timestamp | Time when resource pool monitoring information is persistently stored |
| nodegroup | Name | Name of the logical cluster the resource pool belongs to. The default cluster is installation . |
| rpname | Name | Resource pool name |
| cgroup | Name | Name of the Cgroup associated with the resource pool |
| ref_count | Int | Number of jobs referenced by the resource pool. The number is counted regardless of whether the jobs are controlled by the resource pool. This parameter is valid only on CNs. |
| fast_run | Int | Number of running jobs in the fast lane of the resource pool. This parameter is valid only on CNs. |
| fast_wait | Int | Number of jobs queued in the fast lane of the resource pool. This parameter is valid only on CNs. |

| Column | Type | Description |
|-----------------|--------|--|
| fast_limit | Int | Limit on the number of concurrent jobs in the fast lane in a resource pool. This parameter is valid only on CNs. |
| slow_run | Int | Number of running jobs in the slow lane of the resource pool. This parameter is valid only on CNs. |
| slow_wait | Int | Number of jobs queued in the slow lane of the resource pool. This parameter is valid only on CNs. |
| slow_limit | Int | Limit on the number of concurrent jobs in the slow lane in a resource pool. This parameter is valid only on CNs. |
| used_cpu | Double | <p>Average number of CPUs used by the resource pool in a 5s monitoring period. The value is accurate to two decimal places.</p> <ul style="list-style-type: none">On a DN, it indicates the number of CPUs used by the resource pool on the current DN.On a CN, it indicates the total CPU usage of resource pools on all DNs. |
| cpu_limit | Int | <p>It indicates the upper limit of available CPUs for resource pools. If the CPU share is limited, this parameter indicates the available CPUs for DWS. If the CPU limit is specified, this parameter indicates the available CPUs for associated Cgroups.</p> <ul style="list-style-type: none">On a DN, it indicates the upper limit of available CPUs for the resource pool on the current DN.On a CN, it indicates the total upper limit of available CPUs for resource pools on all DNs. |
| used_mem | Int | <p>Memory used by the resource pool, in MB</p> <ul style="list-style-type: none">On a DN, it indicates the memory usage of the resource pool on the current DN.On a CN, it indicates the total memory usage of resource pools on all DNs. |
| estimate_memory | Int | Estimated memory used by the jobs running in the resource pools on the current CN. This parameter is valid only on CNs. |

| Column | Type | Description |
|--------------|--------|---|
| mem_limit | Int | Upper limit of available memory for the resource pool, in MB <ul style="list-style-type: none">On a DN, it indicates the upper limit of available memory for the resource pool on the current DN.On a CN, it indicates the total upper limit of available memory for resource pools on all DNs. |
| read_kbytes | Bigint | Number of logical read bytes in the resource pool within a 5s monitoring period, in KB <ul style="list-style-type: none">On a DN, it indicates the number of logical read bytes in the resource pool on the current DN.On a CN, it indicates the total logical read bytes of resource pools on all DNs. |
| write_kbytes | Bigint | Number of logical write bytes in the resource pool within a 5s monitoring period, in KB <ul style="list-style-type: none">On a DN, it indicates the number of logical write bytes in the resource pool on the current DN.On a CN, it indicates the total logical write bytes of resource pools on all DNs. |
| read_counts | Bigint | Number of logical reads in the resource pool within a 5s monitoring period <ul style="list-style-type: none">On a DN, it indicates the number of logical reads in the resource pool on the current DN.On a CN, it indicates the total number of logical reads in resource pools on all DNs. |
| write_counts | Bigint | Number of logical writes in the resource pool within a 5s monitoring period. <ul style="list-style-type: none">On a DN, it indicates the number of logical writes in the resource pool on the current DN.On a CN, it indicates the total number of logical writes in resource pools on all DNs. |
| read_speed | Double | Average logical read rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the logical read rate of the resource pool on the current DN.On a CN, it indicates the overall logical read rate of resource pools on all DNs. |

| Column | Type | Description |
|-------------|--------|---|
| write_speed | Double | Average logical write rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the logical write rate of the resource pool on the current DN.On a CN, it indicates the overall logical write rate of resource pools on all DNs. |
| send_speed | Double | Average network sending rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the network sending rate of the resource pool on the current DN.On a CN, it indicates the sum of the network sending rates of the resource pool on all DNs. |
| recv_speed | Double | Average network receiving rate of a resource pool in a 5-second monitoring period, in KB/s <ul style="list-style-type: none">On a DN, it indicates the network receiving rate of the resource pool on the current DN.On a CN, it indicates the sum of the network receiving rates of the resource pool on all DNs. |

18.3.218 PGXC_ROW_TABLE_IO_STAT

PGXC_ROW_TABLE_IO_STAT provides I/O statistics of all row-store tables of the database on all CNs and DNs in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_ROW_TABLE_IO_STAT** view. For details about the columns, see [Table 18-292](#).

Table 18-292 GS_ROW_TABLE_IO_STAT columns

| Column | Type | Description |
|------------|--------|---|
| schemaname | Name | Namespace of a table |
| relname | Name | Name of a table |
| heap_read | Bigint | Number of blocks logically read in the heap |
| heap_hit | Bigint | Number of block hits in the heap |
| idx_read | Bigint | Number of blocks logically read in the index |
| idx_hit | Bigint | Number of block hits in the index |
| toast_read | Bigint | Number of blocks logically read in the TOAST table |

| Column | Type | Description |
|-----------|--------|--|
| toast_hit | Bigint | Number of block hits in the TOAST table |
| tidx_read | Bigint | Number of indexes logically read in the TOAST table |
| tidx_hit | Bigint | Number of index hits in the TOAST table |

18.3.219 PGXC_RUNNING_XACTS

PGXC_RUNNING_XACTS displays information about running transactions on each node in the cluster. The content is the same as that displayed in [PG_RUNNING_XACTS](#).

Table 18-293 PGXC_RUNNING_XACTS columns

| Column | Type | Description |
|-------------|---------|---|
| handle | Integer | Handle corresponding to the transaction in GTM |
| gxid | Xid | Transaction ID |
| state | tinyint | Transaction status (3 : prepared or 0 : starting) |
| node | Text | Node name |
| xmin | Xid | Minimum transaction ID xmin on the node |
| vacuum | boolean | Whether the current transaction is lazy vacuum |
| timeline | Bigint | Number of database restarts |
| prepare_xid | Xid | Transaction ID in prepared state. If the status is not prepared , the value is 0 . |
| pid | Bigint | Thread ID corresponding to the transaction |
| next_xid | Xid | Transaction ID sent from a CN to a DN |

18.3.220 PGXC_SETTINGS

PGXC_SETTINGS displays the database running status of each node in the cluster. All columns except **node_name** are the same as those in the [PG_SETTINGS](#) view. This view is accessible only to users with system administrators rights.

Table 18-294 PGXC_SETTINGS columns

| Column | Type | Description |
|------------|---------|---|
| node_name | Text | Node name |
| Name | Text | Parameter name |
| setting | Text | Current value of the parameter |
| unit | Text | Implicit unit of the parameter |
| category | Text | Logical group of the parameter |
| short_desc | Text | Brief description of the parameter |
| extra_desc | Text | Detailed description of the parameter |
| context | Text | Context of parameter values including internal , postmaster , sighup , backend , superuser , and user . |
| vartype | Text | Parameter type. It can be bool , enum , integer , real , or string . |
| source | Text | Method of assigning the parameter value |
| min_val | Text | Minimum value of the parameter. If the parameter type is not numeric data, the value of this column is null. |
| max_val | Text | Maximum value of the parameter. If the parameter type is not numeric data, the value of this column is null. |
| enumvals | Text[] | Valid values of an enum-typed parameter. If the parameter type is not enum, the value of this column is null. |
| boot_val | Text | Default parameter value used upon the database startup |
| reset_val | Text | Default parameter value used upon the database reset |
| sourcefile | Text | Configuration file used to set parameter values. If parameter values are not configured using the configuration file, the value of this column is null. |
| sourceline | Integer | Row number of the configuration file for setting parameter values. If parameter values are not configured using the configuration file, the value of this column is null. |

18.3.221 PGXC_SESSION_WLMSTAT

PGXC_SESSION_WLMSTAT displays load management information about ongoing jobs executed on each CN in the current cluster.

Table 18-295 PGXC_SESSION_WLMSTAT columns

| Column | Type | Description |
|------------------|---------|--|
| nodename | Name | Node name. |
| datid | OID | OID of the database the backend is connected to. |
| datname | Name | Name of the database the backend is connected to. |
| threadid | Bigint | ID of the backend thread. |
| processid | Integer | PID of a backend thread |
| usesysid | OID | OID of the user who logged in to the backend |
| appname | Text | Name of the application that is connected to the backend |
| username | Name | Name of the user logged in to the backend |
| priority | Bigint | Priority of Cgroup where the statement is located |
| attribute | Text | Statement attributes <ul style="list-style-type: none">• Ordinary: default attribute of a statement before it is parsed by the database• Simple: simple statements• Complicated: complicated statements• Internal: internal statement of the database |
| block_time | Bigint | Pending duration of the statements by now (unit: s) |
| elapsed_time | Bigint | Actual execution duration of the statements by now (unit: s) |
| total_cpu_time | Bigint | Total CPU usage duration of the statement on the DN in the last period (unit: s) |
| cpu_skew_percent | Integer | CPU usage inclination ratio of the statement on the DN in the last period |
| statement_mem | Integer | Estimated memory required for statement execution. This column is reserved. |
| active_points | Integer | Number of concurrently active points occupied by the statement in the resource pool |
| dop_value | Integer | DOP value obtained by the statement from the resource pool |

| Column | Type | Description |
|---------------|------|--|
| control_group | Text | Cgroup currently used by the statement |
| status | Text | Status of a statement, including: <ul style="list-style-type: none">• pending• running: The statement is being executed.• finished: The execution is finished normally. (If enqueue is set to StoredProc or Transaction, this state indicates that only some of the jobs in the statement have been executed. This state persists until the finish of this statement.)• aborted: terminated unexpectedly• active: normal status except for those above• unknown: unknown status |
| enqueue | Text | Current queuing status of the statements, including: <ul style="list-style-type: none">• Global: global queuing.• Respool: resource pool queuing.• CentralQueue: queuing on the CCN• Transaction: The statements are in the same transaction block.• StoredProc: The statement is in a stored procedure.• None: not in a queue• Forced None: The transaction block statement or stored procedure statement is being forcibly executed because the statement waiting time exceeds the specified value. |
| resource_pool | Name | Current resource pool where the statements are located. |
| query | Text | Text of this backend's most recent query. If state is active , this column shows the executing query. In all other states, it shows the last query that was executed. |
| isplana | Bool | In logical cluster mode, indicates whether a statement occupies the resources of other logical clusters. The default value is f , indicating that resources of other logical clusters are not occupied. |
| node_group | Text | Logical cluster of the user running the statement |

| Column | Type | Description |
|--------|------|--|
| lane | Text | Fast or slow lane for statement queries. <ul style="list-style-type: none">• fast: fast lane• slow: slow lane• none: not controlled |

18.3.222 PGXC_STAT_ACTIVITY

PGXC_STAT_ACTIVITY displays information about the query performed by the current user on all the CNs in the current cluster.

Table 18-296 PGXC_STAT_ACTIVITY columns

| Column | Type | Description |
|------------------|---------|---|
| coorname | Text | Name of the CN in the current cluster |
| datid | OID | OID of the database that the user session connects to in the backend |
| datname | Name | Name of the database that the user session connects to in the backend |
| pid | Bigint | ID of the backend thread |
| lwtid | Integer | Lightweight thread ID of the backend thread |
| usesysid | OID | OID of the user logging in to the backend |
| username | Name | Name of the user logging in to the backend |
| application_name | Text | Name of the application connected to the backend |
| client_addr | inet | IP address of the client connected to the backend. If this column is null , it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |

| Column | Type | Description |
|---------------|--------------------------|---|
| client_port | Integer | TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used |
| backend_start | Timestamp with time zone | Startup time of the backend process, that is, the time when the client connects to the server |
| xact_start | Timestamp with time zone | Time when the current transaction was started, or NULL if no transaction is active. If the current query is the first of its transaction, this column is equal to the query_start column. |
| query_start | Timestamp with time zone | Time when the currently active query was started, or time when the last query was started if state is not active |
| state_change | Timestamp with time zone | Time for the last status change |
| waiting | boolean | The value is t if the backend is waiting for a lock or node. Otherwise, the value is f . |

| Column | Type | Description |
|---------|------|--|
| enqueue | Text | <p>Queuing status of a statement. Its value can be:</p> <ul style="list-style-type: none">• waiting in global queue: The statement is in the global concurrent queues.• waiting in respool queue: The statement is queuing in the resource pool. The scenarios are as follows:<ol style="list-style-type: none">1. When dynamic load balancing is enabled, the number of simple jobs exceeds the upper limit (max_dop) of concurrent jobs on the fast lane.2. When dynamic load balancing is disabled, the number of simple jobs exceeds the upper limit (max_dop) of concurrent jobs on the fast lane or the number of complex jobs exceeds the upper limit of concurrent jobs on the slow lane.• waiting in ccn queue: The job is in the CCN queue, which may be global memory queuing, slow lane memory queuing, or concurrent queuing.• Empty or no waiting queue: The statement is running. |

| Column | Type | Description |
|-----------------|--------|--|
| state | Text | <p>Overall state of the backend. Its value can be:</p> <ul style="list-style-type: none">• active: The backend is executing a query.• idle: The backend is waiting for a new client command.• idle in transaction: The backend is in a transaction, but there is no statement being executed in the transaction.• idle in transaction (aborted): The backend is in a transaction, but there are statements failed in the transaction.• fastpath function call: The backend is executing a fast-path function.• disabled: This state is reported if track_activities is disabled in this backend. <p>NOTE
Only system administrators can view the session status of their accounts. The state information of other accounts is empty.</p> |
| resource_pool | Name | Resource pool used by the user |
| stmt_type | Text | Type of a user statement |
| query_id | Bigint | ID of a query |
| query | Text | Text of this backend's most recent query
If the state is active , this column shows the executing query. In all other states, it shows the last query that was executed. |
| connection_info | Text | A string in JSON format recording the driver type, driver version, driver deployment path, and process owner of the connected database (for details, see connection_info). |

Example

Run the following command to view blocked query statements.

```
SELECT datname,username,state,query FROM PGXC_STAT_ACTIVITY WHERE waiting = true;
```

Check the working status of the snapshot thread.

```
SELECT application_name,backend_start,state_change,state,query FROM PGXC_STAT_ACTIVITY WHERE  
application_name='WDRSnapshot';
```

View the running query statements.

```
SELECT datname,username,state,pid FROM PGXC_STAT_ACTIVITY;  
datname | username | state | pid  
-----+-----+-----+-----  
gaussdb | Ruby    | active | 140298793514752  
gaussdb | Ruby    | active | 140298718004992  
gaussdb | Ruby    | idle   | 140298650908416  
gaussdb | Ruby    | idle   | 140298625742592  
gaussdb | dbadmin | active | 140298575406848  
(5 rows)
```

View the number of session connections that have been used by postgres. **1** indicates the number of session connections that have been used by **postgres**.

```
SELECT COUNT(*) FROM PGXC_STAT_ACTIVITY WHERE DATNAME='postgres';  
count  
-----  
1  
(1 row)
```

18.3.223 PGXC_STAT_BAD_BLOCK

PGXC_STAT_BAD_BLOCK displays statistics about page or CU verification failures after all nodes in a cluster are started.

Table 18-297 PGXC_STAT_BAD_BLOCK columns

| Column | Type | Description |
|--------------|--------------------------|----------------------------------|
| nodename | Text | Node name. |
| databaseid | Integer | Database OID. |
| tablespaceid | Integer | Tablespace OID. |
| relfilenode | Integer | File object ID. |
| forknum | Integer | File type. |
| error_count | Integer | Number of verification failures. |
| first_time | Timestamp with time zone | Time of the first occurrence. |
| last_time | Timestamp with time zone | Time of the latest occurrence. |

18.3.224 PGXC_STAT_BGWRITER

PGXC_STAT_BGWRITER displays statistics on the background writer of each node in the cluster. All columns except **node_name** are the same as those in the [PG_STAT_BGWRITER](#) view. This view is accessible only to users with system administrators rights.

Table 18-298 PGXC_STAT_BGWRITER columns

| Column | Type | Description |
|-----------------------|--------------------------|---|
| node_name | Text | Node name |
| checkpoints_timed | Bigint | Number of scheduled checkpoints that have been performed |
| checkpoints_req | Bigint | Number of requested checkpoints that have been performed |
| checkpoint_write_time | Double precision | Time spent on writing files to the disk during checkpoints, in milliseconds |
| checkpoint_sync_time | Double precision | Time spent on synchronizing data to the disk during checkpoints, in milliseconds |
| buffers_checkpoint | Bigint | Number of buffers written during checkpoints |
| buffers_clean | Bigint | Number of buffers written by the background writer |
| maxwritten_clean | Bigint | Number of times the background writer stopped a cleaning scan because it had written too many buffers |
| buffers_backend | Bigint | Number of buffers written directly by a backend |
| buffers_backend_fsync | Bigint | Number of times that a backend has to execute fsync |
| buffers_alloc | Bigint | Number of buffers allocated |
| stats_reset | Timestamp with time zone | Time at which these statistics were reset |

18.3.225 PGXC_STAT_DATABASE

PGXC_STAT_DATABASE displays the database status and statistics of each node in the cluster. All columns except **node_name** are the same as those in the **PG_STAT_DATABASE** view. This view is accessible only to users with system administrators rights.

Table 18-299 PGXC_STAT_DATABASE columns

| Column | Type | Description |
|-----------|------|--------------|
| node_name | Text | Node name |
| datid | OID | Database OID |

| Column | Type | Description |
|---------------|---------|---|
| datname | Name | Database name |
| numbackends | Integer | Number of backends currently connected to this database on the current node. This is the only column in this view that reflects the current state value. All columns return the accumulated value since the last reset. |
| xact_commit | Bigint | Number of transactions in this database that have been committed on the current node |
| xact_rollback | Bigint | Number of transactions in this database that have been rolled back on the current node |
| blks_read | Bigint | Number of disk blocks read in this database on the current node |
| blks_hit | Bigint | Number of disk blocks found in the buffer cache on the current node, that is, the number of blocks hit in the cache. (This only includes hits in the DWS buffer cache, not in the file system cache.) |
| tup_returned | Bigint | Number of rows returned by queries in this database on the current node |
| tup_fetched | Bigint | Number of rows fetched by queries in this database on the current node |
| tup_inserted | Bigint | Number of rows inserted in this database on the current node |
| tup_updated | Bigint | Number of rows updated in this database on the current node |
| tup_deleted | Bigint | Number of rows deleted from this database on the current node |
| conflicts | Bigint | Number of queries canceled due to database recovery conflicts on the current node (conflicts occurring only on the standby server). For details, see PG_STAT_DATABASE_CONFLICTS . |
| temp_files | Bigint | Number of temporary files created by this database on the current node. All temporary files are counted, regardless of why the temporary file was created (for example, sorting or hashing), and regardless of the log_temp_files setting. |

| Column | Type | Description |
|----------------|--------------------------|---|
| temp_bytes | Bigint | Size of temporary files written to this database on the current node. All temporary files are counted, regardless of why the temporary file was created, and regardless of the log_temp_files setting. |
| deadlocks | Bigint | Number of deadlocks in this database on the current node |
| blk_read_time | Double precision | Time spent reading data file blocks by backends in this database on the current node, in milliseconds |
| blk_write_time | Double precision | Time spent writing into data file blocks by backends in this database on the current node, in milliseconds |
| stats_reset | Timestamp with time zone | Time when the database statistics are reset on the current node |

18.3.226 PGXC_STAT_OBJECT

PGXC_STAT_OBJECT displays statistics and autovacuum efficiency information about tables of all instances in a cluster. This system view is supported only by clusters of version 8.2.1 or later.

Table 18-300 PGXC_STAT_OBJECT columns

| Column | Type | Reference | Description |
|--------------|------|--------------------------------------|--|
| nodename | Name | - | Node name |
| datname | Name | - | Name of the database where the table is located. |
| relnamespace | Name | - | Name of the schema where the table is located. |
| relname | Name | - | Table name. |
| partname | Name | - | Partition name of the partitioned table |
| databaseid | OID | PG_DATA
BASE.oid | Database OID. |
| relid | OID | PG_CLAS
S.oid | Table OID. It is the OID of the primary table for a partitioned table. |

| Column | Type | Reference | Description |
|------------------------------------|--------------------------|----------------------------------|---|
| partid | OID | PG_PARTITION.oid | Partition OID. If the table is not partitioned, the value is 0 . |
| numscans | Bigint | - | Number of times that sequential scans are started. |
| tuples_returned | Bigint | - | Number of visible tuples fetched by sequential scans. |
| tuples_fetched | Bigint | - | Number of visible tuples fetched. |
| tuples_inserted | Bigint | - | Number of inserted records. |
| tuples_updated | Bigint | - | Number of updated records. |
| tuples_deleted | Bigint | - | Number of deleted records. |
| tuples_hot_updated | Bigint | - | Number of HOT updates. |
| n_live_tuples | Bigint | - | Number of visible tuples. |
| last_autovacuum_begin_n_dead_tuple | Bigint | - | Number of tuples deleted before Autovacuum is executed. |
| n_dead_tuples | Bigint | - | Number of tuples deleted after Autovacuum is successful. |
| changes_since_analyze | Bigint | - | Last data modification time after Analyze. |
| blocks_fetched | Bigint | - | Number of selected pages. |
| blocks_hit | Bigint | - | Number of scanned pages. |
| cu_mem_hit | Bigint | - | Number of CU memory hits. |
| cu_hdd_sync | Bigint | - | Times that CUs are synchronously read from disks. |
| cu_hdd_async | Bigint | - | Times that CUs are asynchronously read from disks. |
| data_changed_timestamp | Timestamp with time zone | - | Last data modification time. |

| Column | Type | Reference | Description |
|------------------------------|--------------------------|-----------|--|
| data_access_timestamp | Timestamp with time zone | - | Last access time of a table. |
| analyze_timestamp | Timestamp with time zone | - | Last Analyze time. |
| analyze_count | Bigint | - | Total number of Analyze times. |
| autovac_analyze_timestamp | Timestamp with time zone | - | Last Autoanalyze time. |
| autovac_analyze_count | Bigint | - | Total number of Autoanalyze times. |
| vacuum_timestamp | Timestamp with time zone | - | Time of the latest Vacuum. |
| vacuum_count | Bigint | - | Total number of Vacuum times. |
| autovac_vacuum_timestamp | Timestamp with time zone | - | Last Autovacuum time. |
| autovac_vacuum_count | Bigint | - | Total number of Autovacuum times. |
| autovacuum_success_count | Bigint | - | Total number of successful Autovacuum operations. |
| last_autovacuum_time_cost | Bigint | - | Time spent on the latest successful Autovacuum, in microseconds. |
| avg_autovacuum_time_cost | Bigint | - | Average execution time of successful Autovacuum operations. Unit: μ s. |
| last_autovacuum_failed_count | Bigint | - | Total number of autovacuum failures since the last successful Autovacuum. |
| last_autovacuum_trigger | Smallint | - | Triggering mode of the latest autovacuum, which helps maintenance personnel determine the Vacuum status. |

| Column | Type | Reference | Description |
|----------------------------------|--------|-----------|---|
| last_autovacuum_oldestxmin | Bigint | - | oldestxmin after the latest successful Autovacuum execution. If the table-level oldestxmin feature is enabled, this field records the value of oldestxmin used by the latest (AUTO)VACUUM of the table. |
| last_autovacuum_scan_pages | Bigint | - | Number of pages last scanned by autovacuum (only for row-store tables). |
| last_autovacuum_dirty_pages | Bigint | - | Number of pages last modified by Autovacuum (only for row-store tables). |
| last_autovacuum_clear_deadtuples | Bigint | - | Number of dead tuples last cleared by Autovacuum (only for row-store tables) |
| sum_autovacuum_scan_pages | Bigint | - | Total number of pages scanned by Autovacuum since database initialization (only for row-store tables). |
| sum_autovacuum_dirty_pages | Bigint | - | Number of pages modified by Autovacuum since database initialization (only for row-store tables). |
| sum_autovacuum_clear_deadtuples | Bigint | - | Total number of dead tuples cleared by Autovacuum since database initialization (only for row-store tables). |
| last_autovacuum_begin_cu_size | Bigint | - | Size of the CU file before the latest Autovacuum operation (only for column-store tables). |
| last_autovacuum_cu_size | Bigint | - | Size of the CU file after the latest Autovacuum (only for column-store tables). |
| last_autovacuum_rewrite_size | Bigint | - | Size of the column-store file last rewritten by autovacuum (only for column-store tables). |
| last_autovacuum_clear_size | Bigint | - | Size of the column-store file last cleared by Autovacuum (only for column-store tables). |

| Column | Type | Reference | Description |
|-------------------------------------|--------------------------|-----------|---|
| last_autovacuum_clear_cbtree_tuples | Bigint | - | Number of cbtree tuples last cleared by Autovacuum (only for column-store tables). |
| sum_autovacuum_rewrite_size | Bigint | - | Total size of column-store files rewritten by Autovacuum since database initialization (only for column-store tables). |
| sum_autovacuum_clear_size | Bigint | - | Total size of column-store files cleared by Autovacuum since database initialization (only for column-store tables). |
| sum_autovacuum_clear_cbtree_tuples | Bigint | - | Total number of cbtree tuples cleared by Autovacuum since database initialization (only for column-store tables). |
| last_autovacuum_csn | Bigint | - | If the table-level oldestxmin feature is enabled, this field records the CSN value corresponding to the latest oldestxmin value used by the table (AUTO)VACUUM . |
| last_reference_timestamp | Timestamp with time zone | - | Last access time of a table. (This field is supported only by cluster versions 8.3.0 and later.)

This parameter corresponds to the latest time between data_changed_time_stamp (last modification time) and data_access_timestamp (last access time) in PG_STAT_OBJECT . |
| extra1 | Bigint | - | Reserved field 1. |
| extra2 | Bigint | - | Reserved field 2. |
| extra3 | Bigint | - | Reserved field 3. |
| extra4 | Bigint | - | Reserved field 4. |

18.3.227 PGXC_STAT_REPLICATION

PGXC_STAT_REPLICATION displays the log synchronization status of each node in the cluster. All columns except **node_name** are the same as those in the [PG_STAT_REPLICATION](#) view. This view is accessible only to users with system administrators rights.

Table 18-301 PGXC_STAT_REPLICATION columns

| Column | Type | Description |
|--------------------------|--------------------------|---|
| node_name | Text | Node name |
| pid | Bigint | PID of the thread |
| usesysid | OID | User system ID |
| username | Name | Username |
| application_name | Text | Program name |
| client_addr | inet | Client address |
| client_hostname | Text | Client name |
| client_port | Integer | Client port number |
| backend_start | Timestamp with time zone | Program start time |
| state | Text | Log replication state (catch-up or consistent streaming) |
| sender_sent_location | Text | Location where the sender sends logs |
| receiver_write_location | Text | Location where the receiver writes logs |
| receiver_flush_location | Text | Location where the receiver flushes logs |
| receiver_replay_location | Text | Location where the receiver replays logs |
| sync_priority | Integer | Priority of synchronous duplication (0 indicates asynchronization) |
| sync_state | Text | Synchronization state (asynchronous duplication, synchronous duplication, or potential synchronization) |

18.3.228 PGXC_STAT_TABLE_DIRTY

PGXC_STAT_TABLE_DIRTY displays statistics about all the tables on all the CNs and DN in the current cluster, and the dirty page rate of tables on a single CN or DN. This view is supported only by clusters of version 8.1.3 or later.

 NOTE

The statistics of this view depend on the **ANALYZE** operation. To obtain the most accurate information, perform the **ANALYZE** operation on the table first.

Table 18-302 PGXC_STAT_TABLE_DIRTY columns

| Column | Type | Description |
|-------------------|-------------------------|--|
| nodename | Text | Node name |
| schema | Name | Schema name of the table |
| tablename | Name | Table name |
| partname | Name | Partition name of the partitioned table |
| last_vacuum | timestampwith time zone | Time of the last manual VACUUM |
| last_autovacuum | timestampwith time zone | Time of the last AUTOVACUUM |
| last_analyze | timestampwith time zone | Time when the last non-automatic rolling analysis is performed. |
| last_autoanalyze | timestampwith time zone | Time when the last automatic rolling analysis is performed. |
| vacuum_count | Bigint | Number of times VACUUM operations |
| autovacuum_count | Bigint | Number of AUTOVACUUM operations |
| analyze_count | Bigint | Number of ANALYZE operations |
| autoanalyze_count | Bigint | Number of times of automatically performing the ANALYZE operation |
| n_tup_ins | Bigint | Number of rows inserted |
| n_tup_upd | Bigint | Number of rows updated |
| n_tup_del | Bigint | Number of rows deleted |
| n_tup_hot_upd | Bigint | Number of rows with HOT updates |
| n_tup_change | Bigint | Number of changed rows after ANALYZE |
| n_live_tup | Bigint | Estimated number of live rows |
| n_dead_tup | Bigint | Estimated number of dead rows |

| Column | Type | Description |
|-------------------|-------------------------|---|
| dirty_rate | Bigint | Dirty page rate of a single CN or DN |
| last_data_changed | timestampwith time zone | Time when a table was last modified

This column is left empty in cluster 8.2.0 and later versions. For details, see PG_STAT_ALL_TABLES . |

Suggestion

- Before running **VACUUM FULL** on a system catalog with a high dirty page rate, ensure that no user is performing operations on it.
- You are advised to run **VACUUM FULL** to tables (excluding system catalogs) whose dirty page rate exceeds 80% or run it based on service scenarios.

Scenarios

1. Query the overall dirty page rate of all the user tables in a database.

```
select
  t1.schema,
  t1.tablename,
  t1.total_ins,
  t1.total_upd,
  t1.total_del,
  t1.total_tup_hot_upd,
  t1.total_change,
  t1.total_live,
  t1.total_dead,
  t1.total_dirty_rate,
  t1.max_dirty,
  t2.max_node,
  t1.min_dirty,
  t2.min_node
from
  (select
    a.schema,
    a.tablename,
    sum(a.n_tup_ins) as total_ins,
    sum(a.n_tup_upd) as total_upd,
    sum(a.n_tup_del) as total_del,
    sum(a.n_tup_hot_upd) as total_tup_hot_upd,
    sum(a.n_tup_change) as total_change,
    sum(a.n_live_tup) as total_live,
    sum(a.n_dead_tup) as total_dead,
    Round((total_dead / (total_dead + total_live + 0.0001) * 100),2) AS total_dirty_rate,
    max(a.dirty_rate) as max_dirty,
    min(a.dirty_rate) as min_dirty
    from pg_catalog.pgxc_stat_table_dirty a where a.partname is null and a.schema not in
    ('pg_toast','cstore','gs_logical_cluster','sys','dbms_om','information_schema','pg_catalog','dbms_output',
    'dbms_random','utl_raw','utl_raw dbms_sql','dbms_lob') group by a.tablename, a.schema
  ) t1,
  (select distinct
    tablename, schema,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate) as min_node,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate desc) as max_node
    from (select * from pg_catalog.pgxc_stat_table_dirty)) t2
where t1.tablename = t2.tablename and t1.schema = t2.schema;
```

2. Query the overall dirty page rate of all the tables (user tables and system catalogs) in a database.

```
select
  t1.schema,
  t1.tablename,
  t1.total_ins,
  t1.total_upd,
  t1.total_del,
  t1.total_tup_hot_upd,
  t1.total_change,
  t1.total_live,
  t1.total_dead,
  t1.total_dirty_rate,
  t1.max_dirty,
  t2.max_node,
  t1.min_dirty,
  t2.min_node
from
  (select
    a.schema,
    a.tablename,
    sum(a.n_tup_ins) as total_ins,
    sum(a.n_tup_upd) as total_upd,
    sum(a.n_tup_del) as total_del,
    sum(a.n_tup_hot_upd) as total_tup_hot_upd,
    sum(a.n_tup_change) as total_change,
    sum(a.n_live_tup) as total_live,
    sum(a.n_dead_tup) as total_dead,
    Round((total_dead / (total_dead + total_live + 0.0001) * 100),2) AS total_dirty_rate,
    max(a.dirty_rate) as max_dirty,
    min(a.dirty_rate) as min_dirty
  from pg_catalog.pgxc_stat_table_dirty a where a.partname is null group by a.tablename, a.schema
  ) t1,
  (select distinct
    tablename, schema,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate) as min_node,
    first_value(nodename) over(partition by tablename, schema order by dirty_rate desc) as max_node
  from (select * from pg_catalog.pgxc_stat_table_dirty)) t2
where t1.tablename = t2.tablename and t1.schema = t2.schema;
```

3. Query all system catalogs in a database.

```
select * from pgxc_stat_table_dirty where schema in
('pg_toast','cstore','gs_logical_cluster','sys','dbms_om','information_schema','pg_catalog','dbms_output',
dbms_random','utl_raw','utl_raw dbms_sql','dbms_lob');
```

18.3.229 PGXC_STAT_WAL

PGXC_STAT_WAL displays the WAL logs and data page traffic information of the current query. This view is supported only by clusters 8.2.0 and later versions.

Table 18-303 PGXC_STAT_WAL columns

| Column | Type | Description |
|----------------------|-----------|--|
| query_id | Bigint | ID of the current query |
| query_start | Timestamp | Start time of the query |
| global_wal | Bigint | Total number of WAL logs generated by the current query in the cluster, in bytes |
| global_avg_wal_speed | Bigint | Average rate of WAL log generation for the current query in the cluster, in byte/s |

| Column | Type | Description |
|----------------------------|--------|--|
| global_datapage | Bigint | Total size of data pages generated by the current query in the cluster, in bytes |
| global_avg_data_page_speed | Bigint | Average rate of data page generation for the current query in the cluster, in byte/s |
| min_wal_node | Text | Name of the instance group that generates the smallest volume of WAL logs in the current query |
| min_wal | Bigint | Minimum WAL logs generated by a node, in bytes |
| max_wal_node | Text | Name of the instance group that generates the largest volume of WAL logs in the current query |
| max_wal | Bigint | Maximum WAL logs generated by a node, in bytes |
| min_datapage_node | Text | Name of the instance group that generates the smallest volume of data pages in the current query |
| min_data_page | Bigint | Minimum data pages generated by a node, in bytes |
| max_datapage_node | Text | Name of the instance group that generates the largest volume of data pages in the current query |
| max_data_page | Bigint | Maximum data pages generated by a node, in bytes |
| avg_wal_per_node | Bigint | Average WAL logs generated by each node, in bytes |
| avg_datapage_per_node | Bigint | Average data pages generated by each node, in bytes |
| query | Text | Statement that is being executed |

 NOTE

When row-store data is imported in batches without indexes, the Xlogs related to logical new pages are generated during data page copy. If the volume of Xlogs is greater than the default value, flow control will be triggered.

Examples

Query the statements that are being executed in the cluster, the total volumes of WAL logs and data pages generated by these statements, their average generation rates, and their distribution on DN.

```
SELECT * FROM PGXC_STAT_WAL;  
query_id | query_start | global_wal | global_avg_wal_speed | global_datapage |  
global_avg_datapage_speed | min_wal_node | min_wal | max_wal_node | max_wal | min_datapage_node |
```

```

min_data_page | max_datapage_node | max_data_page | avg_wal_per_node |
avg_datapage_per_node | query
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
72620543991351767 | 2022-11-10 16:49:47.743291+08 | 7579052 | 419000 | 284057600
| 15740000 | datanode1 | 7579052 | datanode1 | 7579052 | datanode1 | 284057600
| datanode1 | 284057600 | 7579052 |
284057600 | insert into mpptest3 select * from mpptest3;
72620543991351781 | 2022-11-10 16:50:00.616697+08 | 55022176 | 10638000 | 0
| 0 | datanode1 | 55022176 | datanode1 | 55022176 | datanode1 | 0 |
| datanode1 | 0 | 55022176 |
0 | insert into mpptest1 select * from mpptest1;
(2 rows)

```

18.3.230 PGXC_SQL_COUNT

PGXC_SQL_COUNT displays the node-level and user-level statistics for the SQL statements of **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** and DDL, DML, and DCL statements of each CN in a cluster in real time, identifies query types with heavy load, and measures the capability of a cluster or a node to perform a specific type of query. You can calculate QPS based on the quantities and response time of the preceding types of SQL statements at certain time points. For example, **USER1 SELECT** is counted as **X1** at T1 and as **X2** at T2. The **SELECT** QPS of the user can be calculated as follows: $(X2 - X1)/(T2 - T1)$. In this way, the system can draw cluster-user-level QPS curve graphs and determine cluster throughput, monitoring changes in the service load of each user. If there are drastic changes, the system can locate the specific statement type (such as **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO**). You can also observe QPS curves to determine the time points when problems occur and then locate the problems using other tools. The curves provide a basis for optimizing cluster performance and locating problems.

Columns in the **PGXC_SQL_COUNT** view are the same as those in the **GS_SQL_COUNT** view. For details, see [Table 18-146](#).

NOTE

If a **MERGE INTO** statement can be pushed down and a DN receives it, the statement will be counted on the DN and the value of the **mergeinto_count** column will increment by 1. If the pushdown is not allowed, the DN will receive an **UPDATE** or **INSERT** statement. In this case, the **update_count** or **insert_count** column will increment by 1.

18.3.231 PGXC_TABLE_CHANGE_STAT

PGXC_TABLE_CHANGE_STAT displays the changes of all tables of the database on all CNs in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the [GS_TABLE_CHANGE_STAT](#) view.

Table 18-304 PGXC_TABLE_CHANGE_STAT columns

| Column | Type | Description |
|------------|------|-----------------|
| nodename | Name | Node name |
| schemaname | Name | Table namespace |

| Column | Type | Description |
|-------------------|--------------------------|---|
| relname | Name | Table name |
| last_vacuum | Timestamp with time zone | Time when the last VACUUM operation is performed manually |
| vacuum_count | Bigint | Number of times of manually performing the VACUUM operation |
| last_autovacuum | Timestamp with time zone | Time when the last VACUUM operation is performed automatically |
| autovacuum_count | Bigint | Number of times of automatically performing the VACUUM operation |
| last_analyze | Timestamp with time zone | Time when the last non-automatic rolling analysis is performed. |
| analyze_count | Bigint | Number of times of performing the ANALYZE operation (both manually and automatically) |
| last_autoanalyze | Timestamp with time zone | Time when the last automatic rolling analysis is performed. |
| autoanalyze_count | Bigint | Number of times of automatically performing the ANALYZE operation |
| last_change | Bigint | Time when the last modification (INSERT , UPDATE , or DELETE) is performed |

18.3.232 PGXC_TABLE_STAT

PGXC_TABLE_STAT provides statistics of all tables of the database on all CNs and DN in the cluster. Except the **nodename** column of the name type added in front of each row, the names, types, and sequences of other columns are the same as those in the **GS_TABLE_STAT** view.

Table 18-305 PGXC_TABLE_STAT columns

| Column | Type | Description |
|------------|------|-----------------|
| nodename | Name | Node name |
| schemaname | Name | Table namespace |
| relname | Name | Table name |

| Column | Type | Description |
|-------------------|--------|---|
| seq_scan | Bigint | Number of sequential scans. Only row-store tables are counted. For a partitioned table, the sum of the number of scans of each partition is displayed. |
| seq_tuple_read | Bigint | Number of rows scanned in sequence. Only row-store tables are counted. |
| index_scan | Bigint | Number of index scans. Only row-store tables are counted. |
| index_tuple_read | Bigint | Number of rows scanned by the index. Only row-store tables are counted. |
| tuple_inserted | Bigint | Number of rows inserted |
| tuple_updated | Bigint | Number of rows updated |
| tuple_deleted | Bigint | Number of rows deleted |
| tuple_hot_updated | Bigint | Number of rows with HOT updates. |
| live_tuples | Bigint | Number of live tuples. Query the view on the CN. If ANALYZE is executed, the total number of live tuples in the table is displayed. Otherwise, 0 is displayed. This indicator applies only to row-store tables. |
| dead_tuples | Bigint | Number of dead tuples. Query the view on the CN. If ANALYZE is executed, the total number of dead tuples in the table is displayed. Otherwise, 0 is displayed. This indicator applies only to row-store tables. |

18.3.233 PGXC_THREAD_WAIT_STATUS

PGXC_THREAD_WAIT_STATUS displays all the call layer hierarchy relationship between threads of the SQL statements on all the nodes in a cluster, and the waiting status of the block for each thread, so that you can easily locate the causes of process response failures and similar phenomena.

The definitions of **PGXC_THREAD_WAIT_STATUS** view and **PG_THREAD_WAIT_STATUS** view are the same, because the essence of the **PGXC_THREAD_WAIT_STATUS** view is the query summary result of the **PG_THREAD_WAIT_STATUS** view on each node in the cluster.

Table 18-306 PGXC_THREAD_WAIT_STATUS columns

| Column | Type | Description |
|-----------|------|-------------------|
| node_name | Text | Current node name |

| Column | Type | Description |
|-------------|---------|---|
| db_name | Text | Database name |
| thread_name | Text | Thread name |
| query_id | Bigint | Query ID. It is equivalent to debug_query_id . |
| tid | Bigint | Thread ID of the current thread |
| lwtid | Integer | Lightweight thread ID of the current thread |
| ptid | Integer | Parent thread of the streaming thread |
| tlevel | Integer | Level of the streaming thread |
| smpid | Integer | Concurrent thread ID |
| wait_status | Text | Waiting status of the current thread. For details about the waiting status, see Table 18-239 . |
| wait_event | Text | If wait_status is acquire lock , acquire lwlock , or wait io , this column describes the lock, lightweight lock, and I/O information, respectively. If wait_status is not any of the three values, this column is empty. |

Example:

Assume you run a statement on coordinator1, and no response is returned after a long period of time. In this case, establish another connection to coordinator1 to check the thread status on it.

```
select * from pg_thread_wait_status where query_id > 0;
 node_name | db_name | thread_name | query_id |   tid   |   lwtid | ptid | tlevel | smpid |
 wait_status | wait_event
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 coordinator1 | gaussdb | gsql       | 20971544 | 140274089064208 | 22579 |    |    0 |    0 | wait node:
 datanode4 |
(1 rows)
```

Furthermore, you can view the statement working status on each node in the entire cluster. In the following example, no DNs have threads blocked, and there is a huge amount of data to be read, causing slow execution.

```
select * from pgxc_thread_wait_status where query_id=20971544;
 node_name | db_name | thread_name | query_id |   tid   |   lwtid | ptid | tlevel | smpid |
 wait_status | wait_event
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 datanode1 | gaussdb | coordinator1 | 20971544 | 139902867994384 | 22735 |    |    0 |    0 | wait
 node: datanode3 |
 datanode1 | gaussdb | coordinator1 | 20971544 | 139902838634256 | 22970 | 22735 |    5 |    0 |
 synchronize quit |
 datanode1 | gaussdb | coordinator1 | 20971544 | 139902607947536 | 22972 | 22735 |    5 |    1 |
 synchronize quit |
 datanode2 | gaussdb | coordinator1 | 20971544 | 140632156796688 | 22736 |    |    0 |    0 | wait
 node: datanode3 |
 datanode2 | gaussdb | coordinator1 | 20971544 | 140632030967568 | 22974 | 22736 |    5 |    0 |
```

```
synchronize quit |
datanode2 | gaussdb | coordinator1 | 20971544 | 140632081299216 | 22975 | 22736 | 5 | 1 |
synchronize quit |
datanode3 | gaussdb | coordinator1 | 20971544 | 140323627988752 | 22737 | | 0 | 0 | wait
node: datanode3 |
datanode3 | gaussdb | coordinator1 | 20971544 | 140323523131152 | 22976 | 22737 | 5 | 0 | net
flush data |
datanode3 | gaussdb | coordinator1 | 20971544 | 140323548296976 | 22978 | 22737 | 5 | 1 | net
flush data
datanode4 | gaussdb | coordinator1 | 20971544 | 140103024375568 | 22738 | | 0 | 0 | wait
node: datanode3
datanode4 | gaussdb | coordinator1 | 20971544 | 140102919517968 | 22979 | 22738 | 5 | 0 |
synchronize quit |
datanode4 | gaussdb | coordinator1 | 20971544 | 140102969849616 | 22980 | 22738 | 5 | 1 |
synchronize quit |
coordinator1 | gaussdb | gsql | 20971544 | 140274089064208 | 22579 | | 0 | 0 | wait node:
datanode4 |
(13 rows)
```

18.3.234 PGXC_TOTAL_MEMORY_DETAIL

PGXC_TOTAL_MEMORY_DETAIL displays the memory usage in the cluster. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 18-307 PGXC_TOTAL_MEMORY_DETAIL columns

| Column | Type | Description |
|----------|------|-------------|
| nodename | Text | Node name |

| Column | Type | Description |
|------------|------|---|
| memorytype | Text | <p>Memory name, which can be set to any of the following values:</p> <ul style="list-style-type: none">• max_process_memory: memory used by a DWS cluster instance• process_used_memory: memory used by a DWS process• max_dynamic_memory: maximum dynamic memory• dynamic_used_memory: used dynamic memory• dynamic_peak_memory: dynamic peak value of the memory• dynamic_used_shrctx: maximum dynamic shared memory context• dynamic_peak_shrctx: dynamic peak value of the shared memory context• max_shared_memory: maximum shared memory• shared_used_memory: used shared memory• max_cstore_memory: maximum memory allowed for column store• cstore_used_memory: memory used for column store• max_sctpcomm_memory: maximum memory allowed for the communication library• sctpcomm_used_memory: memory used for the communication library• sctpcomm_peak_memory: memory peak of the communication library• other_used_memory: other used memory• gpu_max_dynamic_memory: maximum GPU memory• gpu_dynamic_used_memory: sum of the available GPU memory and temporary GPU memory• gpu_dynamic_peak_memory: maximum memory used for GPU• pooler_conn_memory: memory used for pooler connections• pooler_freeconn_memory: memory used for idle pooler connections |

| Column | Type | Description |
|-------------|---------|---|
| | | <ul style="list-style-type: none">• storage_compress_memory: memory used for column-store compression and decompression• udf_reserved_memory: memory reserved for the UDF Worker process• mmap_used_memory: memory used for mmap |
| memorybytes | Integer | Size of the used memory (MB) |

18.3.235 PGXC_TOTAL_SCHEMA_INFO

PGXC_TOTAL_SCHEMA_INFO displays the schema space information of all instances in the cluster, providing visibility into the schema space usage of each instance. This view can be queried only on CNs.

Table 18-308 PGXC_TOTAL_SCHEMA_INFO columns

| Column | Type | Description |
|--------------|--------|--------------------|
| schemaname | Text | Schema name. |
| schemaid | OID | Schema OID. |
| databasename | Text | Database name. |
| databaseid | OID | Database OID. |
| nodename | Text | Instance name. |
| nodegroup | Text | Node group name. |
| usedspace | Bigint | Used space size. |
| permspace | Bigint | Space upper limit. |

18.3.236 PGXC_TOTAL_SCHEMA_INFO_ANALYZE

PGXC_TOTAL_SCHEMA_INFO_ANALYZE displays the overall schema space information of the cluster, including the total cluster space, average space of instances, skew ratio, maximum space of a single instance, minimum space of a single instance, and names of the instances with the maximum space and minimum space. It provides visibility into the schema space usage of the entire cluster. This view can be queried only on CNs.

Table 18-309 PGXC_TOTAL_SCHEMA_INFO_ANALYZE columns

| Column | Type | Description |
|--------------|---------|--|
| schemaname | Text | Schema name. |
| databasename | Text | Database name. |
| nodegroup | Text | Node group name. |
| total_value | Bigint | Total cluster space in this schema. |
| avg_value | Bigint | Average space per instance in this schema. |
| skew_percent | Integer | Skew ratio. |
| extend_info | Text | The extended information includes the maximum and minimum space values for a single instance, as well as the names of the instances with the maximum and minimum space values. |

18.3.237 PGXC_TOTAL_USER_RESOURCE_INFO

The **PGXC_TOTAL_USER_RESOURCE_INFO** view displays real-time resource consumption information of users on all instances. This view is supported only by clusters of version 8.2.0 or later.

Table 18-310 PGXC_TOTAL_USER_RESOURCE_INFO columns

| Column | Type | Description |
|-------------|---------|---|
| nodename | Name | Instance name, including CNs and DN. |
| username | Name | Username |
| used_memory | Integer | Used memory (unit: MB)
On a DN, it indicates a user's memory usage on the current DN.
On a CN, it indicates a user's total memory usage on all DNs. |

| Column | Type | Description |
|--------------|------------------|--|
| total_memory | Integer | <p>Available memory (unit: MB). 0 indicates that the available memory is not limited and depends on the maximum memory available in the database.</p> <p>On a DN, it indicates the memory available to a user on the current DN.</p> <p>On a CN, it indicates the total memory available to a user on all DNs.</p> |
| used_cpu | Double precision | <p>Number of CPU cores in use. Only the CPU usage of complex jobs in the non-default resource pool is collected, and the value is the CPU usage of the related cgroup.</p> <p>On a DN, it indicates a user's CPU core usage on the current DN.</p> <p>On a CN, it indicates a user's total CPU core usage on all DNs.</p> |
| total_cpu | Integer | <p>Total number of CPU cores of the Cgroups associated with a user.</p> <p>On a DN, it indicates the CPU cores available to a user on the current DN.</p> <p>On a CN, it indicates the total CPU cores available to a user on all DNs.</p> |
| used_space | Bigint | <p>Used permanent table storage space (unit: KB)</p> <p>On a DN, it indicates the size of the permanent table storage space used by a user on the current DN.</p> <p>On a CN, it indicates the total size of the permanent table storage space used by a user on all DNs.</p> |
| total_space | Bigint | <p>Available storage space (unit: KB). -1 indicates that the storage space is not limited.</p> <p>On a DN, it indicates the size of the permanent table storage space available to a user on the current DN.</p> <p>On a CN, it indicates the total size of the permanent table storage space available to a user on all DNs.</p> |

| Column | Type | Description |
|-------------------|--------|--|
| used_temp_space | Bigint | Used temporary table storage space (unit: KB)
On a DN, it indicates the size of the temporary table storage space used by a user on the current DN.
On a CN, it indicates the total size of the temporary table storage space used by a user on all DNs. |
| total_temp_space | Bigint | Available temporary table storage space (unit: KB). -1 indicates that the storage space is not limited.
On a DN, it indicates the size of the temporary table storage space available to a user on the current DN.
On a CN, it indicates the total size of the temporary table storage space available to a user on all DNs. |
| used_spill_space | Bigint | Size of space used for operator spill to disk, in KB.
On a DN, it indicates the space used by a user to spill operators to disk on the current DN.
On a CN, it indicates the total space used by a user's operators spilled to disk on all DNs. |
| total_spill_space | Bigint | Size of space available for operator spill to disk, in KB. The value -1 indicates that the space is not limited.
On a DN, it indicates the space available for a user to spill operators to disk on the current DN.
On a CN, it indicates the total space available for a user to spill operators to disk on all DNs. |
| read_kbytes | Bigint | On a CN, it indicates the total number of bytes logically read by a user on all DNs in the last 5 seconds, in KB.
On a DN, it indicates the total number of bytes logically read by a user from the instance startup time to the current time, in KB. |
| write_kbytes | Bigint | On a CN, it indicates the total number of bytes logically written by a user on all DNs in the last 5 seconds, in KB.
On a DN, it indicates the total number of bytes logically written by a user from the instance startup time to the current time, in KB. |

| Column | Type | Description |
|--------------|------------------|--|
| read_counts | Bigint | On a CN, it indicates the total number of logical reads performed by a user on all DNs in the last 5 seconds.
On a DN, it indicates the total number of logical reads performed by a user from the instance startup time to the current time. |
| write_counts | Bigint | On a CN, it indicates the total number of logical writes performed by a user on all DNs in the last 5 seconds.
On a DN, it indicates the total number of logical writes performed by a user from the instance startup time to the current time. |
| read_speed | Double precision | On a CN, it indicates the average logical read rate of a user on a single DN in the last 5 seconds, in KB/s.
On a DN, it indicates the average logical read rate of a user on the DN in the last 5 seconds, in KB/s. |
| write_speed | Double precision | On a CN, it indicates the average logical write rate of a user on a single DN in the last 5 seconds, in KB/s.
On a DN, it indicates the average logical write rate of a user on the DN in the last 5 seconds, in KB/s. |
| send_speed | Double precision | On a CN, it indicates the sum of the average network sending rates of a user on all DNs in the last 5 seconds, in KB/s.
On a DN, it indicates the average network sending rate of a user on the DN in the last 5 seconds, in KB/s. |
| recv_speed | Double precision | On a CN, it indicates the sum of the average network receiving rates of a user on all DNs in the last 5 seconds, in KB/s.
On a DN, it indicates the average network receiving rate of a user on the DN in the last 5 seconds, in KB/s. |

18.3.238 PGXC_USER_TRANSACTION

PGXC_USER_TRANSACTION provides transaction information about users on all CNs. It is accessible only to users with system administrator rights. This view is valid only if the real-time resource monitoring function is enabled, that is, if [enable_resource_track](#) is **on**.

Table 18-311 PGXC_USER_TRANSACTION columns

| Column | Type | Description |
|------------------|--------|------------------------|
| node_name | Name | Node name. |
| username | Name | Username. |
| commit_counter | Bigint | Number of commits. |
| rollback_counter | Bigint | Number of rollbacks. |
| resp_min | Bigint | Minimum response time. |
| resp_max | Bigint | Maximum response time. |
| resp_avg | Bigint | Average response time. |
| resp_total | Bigint | Total response time. |

18.3.239 PGXC_VARIABLE_INFO

PGXC_VARIABLE_INFO displays information about XIDs and OIDs of all nodes in a cluster.

Table 18-312 PGXC_VARIABLE_INFO columns

| Column | Type | Description |
|---------------------------|---------|---|
| node_name | Text | Node name. |
| nextOid | OID | Next OID to be generated under this node. |
| nextXid | Xid | Next transaction OID to be generated under this node. |
| oldestXid | Xid | Oldest transaction ID for a node |
| xidVacLimit | Xid | Critical point for forcing autovacuum. |
| oldestXidDB | OID | Database OID with the minimum datafrozenxid under this node. |
| lastExtendCSNL
ogpage | Integer | Page number of the last extension of csnlog. |
| startExtendCSN
Logpage | Integer | Starting page number of the csnlog extension. |
| nextCommitSeq
No | Integer | Next CSN to be generated under this node. |
| latestCompleted
Xid | Xid | Latest transaction ID on the node after commit or rollback. |

| Column | Type | Description |
|---------------|------|---|
| startupMaxXid | Xid | Last transaction ID before the node shutdown. |

18.3.240 PGXC_WAIT_DETAIL

PGXC_WAIT_DETAIL displays detailed information about the SQL waiting hierarchy of all nodes in a cluster. This view is supported only by clusters of version 8.1.3.200 or later.

Table 18-313 PGXC_WAIT_DETAIL columns

| Column | Type | Description |
|---------------------|---------|---|
| level | Integer | Level in the wait hierarchy. The value starts with 1 and increases by 1 when there is a wait relationship. |
| lock_wait_hierarchy | Text | Wait hierarchy, in the format of <i>Node name: Process ID->Node name:Waiting process ID->Node name:Waiting process ID->...</i> |
| node_name | Text | Node name |
| db_name | Text | Database name |
| thread_name | Text | Thread name |
| query_id | Bigint | ID of a query statement |
| tid | Bigint | Thread ID of the current thread |
| lwtid | Integer | Lightweight thread ID of the current thread |
| ptid | Integer | Parent thread of the streaming thread |
| tlevel | Integer | Level of the streaming thread |
| smpid | Integer | Concurrent thread ID |
| wait_status | Text | Waiting status of the current thread |
| wait_event | Text | Virtual ID of the transaction holding or awaiting this lock |
| exec_cn | Boolean | SQL execution CN |
| wait_node | Text | Lock level |
| query | Text | Query statement |
| application_name | Text | Name of the application connected to the backend |

| Column | Type | Description |
|---------------|--------------------------|--|
| backend_start | Timestamp with time zone | Startup time of the backend process, that is, the time when the client connects to the server |
| xact_start | Timestamp with time zone | Start time of the current transaction |
| query_start | Timestamp with time zone | Start time of the active query |
| waiting | Boolean | Waiting status |
| state | Text | Overall state of the backend |
| waittime | Timestamp with time zone | Timestamp when the lock wait starts.
This column is available only in clusters of version 9.1.0.200 or later. |
| holdtime | Timestamp with time zone | Timestamp when the lock starts to be obtained.
This column is available only in clusters of version 9.1.0.200 or later. |

Example

Step 1 Connect to the CN, start a transaction, and perform the update operation.

```
begin;update td set c2=6 where c1=1;
```

Step 2 Open another window to connect to the CN, start another transaction, and perform the update operation. (Do not update the same record concurrently.)

```
begin;update td set c2=6 where c1=7;
```

In this case, the update operation is blocked.

Step 3 Open another window to connect to the CN node and create an index.

```
create index c2_key on td(c2);
```

Step 4 Run the `select * from pgxc_wait_detail;` command.

```
SELECT * FROM PGXC_WAIT_DETAIL;
level | lock_wait_hierarchy          | node_name | db_name | thread_name | query_id
| tid    | lwtid | ptid | tlevel | sm
pid | wait_status | wait_event | exec_cn | wait_node | query            | application_name |
backend_start      | xact_st
art               | query_start   | waiting | state
-----+-----+-----+-----+-----+-----+-----
+-+---+----+---+--+
+-+---+----+---+--+
+-+---+----+---+--+
+-+---+----+---+---
```

| | | | | | | | | |
|---|-----------|-----------------|---|---------|--|----------|-------------------|--|
| 1 | cn_5001: | 139870843444360 | | cn_5001 | postgres | workload | 73183493945299462 | |
| 139870843444360 | 578531 | | 0 | | | | | |
| 0 | wait node | | t | | WLM fetch collect info from data nodes | workload | | |
| <div style="font-family: monospace;">2023-03-13 13:56:56.611486+08 2023-03-14 11:54</div> | | | | | | | | |
| <div style="font-family: monospace;">:33.562808+08 2023-03-13 13:57:00.262736+08 t active</div> | | | | | | | | |
| 1 | cn_5001: | 139870843654544 | | cn_5001 | postgres | qsql | 73183493945299204 | |

```
139870843654544 | 722259 | | 0 |
0 | wait node | | t | | update td set c2=6 where c1=1; | gsql | 2023-03-14
11:52:05.176588+08 | 2023-03-14 11:52
:19.054727+08 | 2023-03-14 11:53:58.114794+08 | t | active
1 | cn_5001:139870843655296 | cn_5001 | postgres | gsql | 73183493945299218 |
139870843655296 | 722301 | | 0 |
0 | wait node | | t | | update td set c2=6 where c1=7; | gsql | 2023-03-14
11:52:08.084265+08 | 2023-03-14 11:52
:42.978132+08 | 2023-03-14 11:53:59.459575+08 | t | active
1 | cn_5001:139870843656424 | cn_5001 | postgres | gsql | 73183493945299223 |
139870843656424 | 722344 | | 0 |
0 | acquire lock | relation | t | | create index c2_key on td(c2); | gsql | 2023-03-14
11:52:10.967028+08 | 2023-03-14 11:52
:53.463227+08 | 2023-03-14 11:54:00.25203+08 | t | active
2 | cn_5001:139870843656424 -> cn_5001:139870843655296 | cn_5001 | postgres | gsql |
73183493945299218 | 139870843655296 | 722344 | |
| | f | | update td set c2=6 where c1=7; | gsql | 2023-03-14
11:52:08.084265+08 | 2023-03-14 11:52
:42.978132+08 | 2023-03-14 11:53:59.459575+08 | t | active
(5 rows)
```

----End

18.3.241 PGXC_WAIT_EVENTS

PGXC_WAIT_EVENTS displays statistics on the waiting status and events of each node in the cluster. The content is the same as that displayed in [GS_WAIT_EVENTS](#). This view is accessible only to users with system administrators rights.

Table 18-314 PGXC_WAIT_EVENTS columns

| Column | Type | Description |
|-----------------|--------|--|
| nodename | Name | Node name. |
| type | Text | Event type, which can be STATUS , LOCK_EVENT , LWLOCK_EVENT , or IO_EVENT |
| event | Text | Event name. For details, see PG_THREAD_WAIT_STATUS . |
| wait | Bigint | Number of times an event occurs. This column and all the columns below are values accumulated during process running. |
| failed_wait | Bigint | Number of waiting failures. In the current version, this column is used only for counting timeout errors and waiting failures of locks such as LOCK and LWLOCK . |
| total_wait_time | Bigint | Total duration of the event |
| avg_wait_time | Bigint | Average duration of the event |
| max_wait_time | Bigint | Maximum wait time of the event |

| Column | Type | Description |
|---------------|--------|--------------------------------|
| min_wait_time | Bigint | Minimum wait time of the event |

18.3.242 PGXC_WLM_OPERATOR_HISTORY

PGXC_WLM_OPERATOR_HISTORY displays operator information when a job is finished on all CNs. This view is used to query data from DWS, and the data in the database is cleared periodically every 3 minutes.

Only the system administrator or the preset role **gs_role_read_all_stats** can access this view. For details about columns in the view, see [Table 18-315](#).

Table 18-315 GS_WLM_OPERATOR_INFO columns

| Column | Type | Description |
|---------------------|--------------------------|---|
| nodename | Text | Name of the CN where the statement is executed |
| queryid | Bigint | Internal query_id used for statement execution |
| pid | Bigint | Backend thread ID |
| plan_node_id | Integer | plan_node_id of the execution plan of a query |
| plan_node_name | Text | Name of the operator corresponding to plan_node_id |
| start_time | Timestamp with time zone | Time when an operator starts to process the first data record |
| duration | Bigint | Total execution time of an operator. The unit is ms. |
| query_dop | Integer | Degree of parallelism (DOP) of the current operator |
| estimated_rows | Bigint | Number of rows estimated by the optimizer |
| tuple_processed | Bigint | Number of elements returned by the current operator |
| min_peak_memory | Integer | Minimum peak memory used by the current operator on all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum peak memory used by the current operator on all DN. The unit is MB. |
| average_peak_memory | Integer | Average peak memory used by the current operator on all DN. The unit is MB. |

| Column | Type | Description |
|---------------------|---------|--|
| memory_skew_percent | Integer | Memory usage skew of the current operator among DNs |
| min_spill_size | Integer | Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_cpu_time | Bigint | Minimum execution time of the operator on all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum execution time of the operator on all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total execution time of the operator on all DNs. The unit is ms. |
| cpu_skew_percent | Integer | Skew of the execution time among DNs. |
| warning | Text | Warning. The following warnings are displayed:
1. Sort/SetOp/HashAgg/HashJoin spill
2. The max spill size exceeds the alarm size 256MB
3. The max broadcast size exceeds the alarm size 100MB
4. Early spill
5. Spill times is greater than 3
6. Spill on memory adaptive
7. Hash table conflict
8. NestLoop in hashjoin |

18.3.243 PGXC_WLM_OPERATOR_INFO

PGXC_WLM_OPERATOR_INFO displays the operator information of completed jobs executed on CNs. The data in this view is obtained from [GS_WLM_OPERATOR_INFO](#).

Only the system administrator or the **gs_role_read_all_stats** role can access this view.

NOTICE

The **PGXC_WLM_OPERATOR_INFO** view can be queried only in the **postgres** database. If the view is queried in other databases, an error is reported.

GS_WLM_OPERATOR_INFO lists the columns in the **PGXC_WLM_OPERATOR_INFO** view.

Table 18-316 GS_WLM_OPERATOR_INFO columns

| Column | Type | Description |
|---------------------|--------------------------|--|
| nodename | Text | Name of the CN where the statement is executed |
| queryid | Bigint | Internal query_id used for statement execution |
| pid | Bigint | Backend thread ID |
| plan_node_id | Integer | plan_node_id of the execution plan of a query |
| plan_node_name | Text | Name of the operator corresponding to plan_node_id |
| start_time | Timestamp with time zone | Time when an operator starts to process the first data record |
| duration | Bigint | Total execution time of an operator. The unit is ms. |
| query_dop | Integer | Degree of parallelism (DOP) of the current operator |
| estimated_rows | Bigint | Number of rows estimated by the optimizer |
| tuple_processed | Bigint | Number of elements returned by the current operator |
| min_peak_memory | Integer | Minimum peak memory used by the current operator on all DNs. The unit is MB. |
| max_peak_memory | Integer | Maximum peak memory used by the current operator on all DNs. The unit is MB. |
| average_peak_memory | Integer | Average peak memory used by the current operator on all DNs. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of the current operator among DNs |

| Column | Type | Description |
|--------------------|---------|--|
| min_spill_size | Integer | Minimum spilled data among all DN's when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DN's when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DN's when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_cpu_time | Bigint | Minimum execution time of the operator on all DN's. The unit is ms. |
| max_cpu_time | Bigint | Maximum execution time of the operator on all DN's. The unit is ms. |
| total_cpu_time | Bigint | Total execution time of the operator on all DN's. The unit is ms. |
| cpu_skew_percent | Integer | Skew of the execution time among DN's. |
| warning | Text | Warning. The following warnings are displayed:
1. Sort/SetOp/HashAgg/HashJoin spill
2. The max spill size exceeds the alarm size 256MB
3. The max broadcast size exceeds the alarm size 100MB
4. Early spill
5. Spill times is greater than 3
6. Spill on memory adaptive
7. Hash table conflict
8. NestLoop in hashjoin |

18.3.244 PGXC_WLM_OPERATOR_STATISTICS

PGXC_WLM_OPERATOR_STATISTICS displays the operator information of jobs being executed on CN's. The system administrator can query job operator information of all users in the cluster, while common users can query only their own job operator information.

[Table 18-160](#) lists the columns in the **PGXC_WLM_OPERATOR_STATISTICS** view.

Table 18-317 GS_WLM_OPERATOR_STATISTICS columns

| Column | Type | Description |
|---------------------|--------------------------|---|
| queryid | Bigint | Internal query_id used for statement execution |
| pid | Bigint | ID of the backend thread |
| plan_node_id | Integer | plan_node_id of the execution plan of a query |
| plan_node_name | Text | Name of the operator corresponding to plan_node_id . The maximum length of the operator name is 127 characters (excluding format characters such as spaces). |
| start_time | Timestamp with time zone | Time when the operator starts to be executed for the first time. |
| duration | Bigint | Total execution time of the operator from the start to the end, in milliseconds. |
| status | Text | Execution status of the current operator. The value can be waiting , running , or finished . |
| query_dop | Integer | DOP of the current operator |
| estimated_rows | Bigint | Number of rows estimated by the optimizer. If the number of returned estimated rows exceeds int64_max, int64_max is displayed. |
| tuple_processed | Bigint | Total number of elements returned by the current operator on all DN. If the estimated number of returned rows exceeds int64_max, int64_max is displayed. |
| min_peak_memory | Integer | Minimum peak memory used by the current operator on all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum peak memory used by the current operator on all DN. The unit is MB. |
| average_peak_memory | Integer | Average peak memory used by the current operator on all DN. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of the current operator among DN |
| min_spill_size | Integer | Minimum logical spilled data among all DN when a spill occurs, in MB. The default value is 0. |
| max_spill_size | Integer | Maximum logical spilled data among all DN when a spill occurs, in MB. The default value is 0. |

| Column | Type | Description |
|--------------------|---------|--|
| average_spill_size | Integer | Average logical spilled data among all DNs when a spill occurs, in MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_cpu_time | Bigint | Minimum execution time of the operator on all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum execution time of the operator on all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total execution time of the operator on all DNs. The unit is ms. |
| cpu_skew_percent | Integer | Skew of the execution time among DNs. |
| warning | Text | Warning. The following warnings are displayed:
1. Sort/SetOp/HashAgg/HashJoin spill
2. The max spill size exceeds the alarm size 256MB
3. The max broadcast size exceeds the alarm size 100MB
4. Early spill
5. Spill times is greater than 3
6. Spill on memory adaptive
7. Hash table conflict
8. NestLoop in hashjoin |
| parent_id | Integer | Parent node ID of the operator node. |
| exec_count | Integer | Maximum number of times that the operator node can be executed on all DNs. |
| progress | Text | Progress information of the operator. For the first operator, it is the overall progress of the job. For other operators, it is the progress of the current operator. |
| min_net_size | Bigint | Minimum network communication data volume (KB) of the operator on all DNs. It mainly applies to network operators. |
| max_net_size | Bigint | Maximum network communication data volume (KB) of the operator on all DNs. It mainly applies to network operators. |

| Column | Type | Description |
|-------------------|--------|--|
| total_net_size | Bigint | Total network communication data volume (KB) of the operator on all DNs. It mainly applies to network operators. |
| min_read_bytes | Bigint | Minimum amount of data read by the operator from disks on all DNs. The unit is KB. |
| max_read_bytes | Bigint | Maximum amount of data read by the operator from disks on all DNs. The unit is KB. |
| total_read_bytes | Bigint | Total amount of data read by the operator from disks on all DNs, in KB. |
| min_write_bytes | Bigint | Minimum amount of data written by the operator to disks on all DNs. The unit is KB. |
| max_write_bytes | Bigint | Maximum amount of data written by the operator to disks on all DNs. The unit is KB. |
| total_write_bytes | Bigint | Total amount of data written by the operator to disks on all DNs, in KB. |

18.3.245 PGXC_WLM_SESSION_INFO

PGXC_WLM_SESSION_INFO displays load management information for completed jobs executed on all CNs. The view information comes from the [GS_WLM_SESSION_INFO](#) system catalog.

NOTICE

The **PGXC_WLM_SESSION_INFO** view can be queried only in the **postgres** database. If the view is queried in other databases, an error is reported.

Table 18-318 PGXC_WLM_SESSION_INFO columns

| Column | Type | Description |
|------------|------|--|
| datid | OID | OID of the database the backend is connected to |
| dbname | Text | Name of the database the backend is connected to |
| schemaname | Text | Schema name |
| nodename | Text | Name of the CN where the statement is run |
| username | Text | User name used for connecting to the backend |

| Column | Type | Description |
|---------------------|--------------------------|--|
| application_name | Text | Name of the application that is connected to the backend |
| client_addr | inet | IP address of the client connected to this backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number used by the client to communicate with the backend. If a Unix socket is used, it is -1 . |
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |
| block_time | Bigint | Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is ms. |
| start_time | Timestamp with time zone | Time when the statement starts to be executed |
| finish_time | Timestamp with time zone | Time when the statement execution ends |
| duration | Bigint | Execution time of a statement. The unit is ms. |
| estimate_total_time | Bigint | Estimated execution time of a statement. The unit is ms. |
| status | Text | Final statement execution status. Its value can be finished (normal) or aborted (abnormal). The statement status here is the execution status of the database server. If the statement is successfully executed on the database server but an error is reported in the result set, the statement status is finished . |
| abort_info | Text | Exception information displayed if the final statement execution status is aborted . |
| resource_pool | Text | Resource pool used by the user |

| Column | Type | Description |
|----------------------|---------|--|
| control_group | Text | Cgroup used by the statement |
| estimate_memory | Integer | Estimated memory used by a statement on a single instance. The unit is MB. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DNs. The unit is MB. |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DNs. The unit is MB. |
| average_peak_memory | Integer | Average memory usage during statement execution. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of a statement among DNs |
| spill_info | Text | Spill information for the statement on all DNs. The options are:
None : The statement has not been spilled to disks on any DNs.
All : The statement has been spilled to disks on all DNs.
[<i>a</i>:<i>b</i>] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DNs. |
| min_spill_size | Integer | Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 . |
| max_spill_size | Integer | Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 . |
| average_spill_size | Integer | Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0 . |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_dn_time | Bigint | Minimum execution time of a statement across all DNs. The unit is ms. |
| max_dn_time | Bigint | Maximum execution time of a statement across all DNs. The unit is ms. |
| average_dn_time | Bigint | Average execution time of a statement across all DNs. The unit is ms. |
| dn_time_skew_percent | Integer | Execution time skew of a statement among DNs. |

| Column | Type | Description |
|-------------------|---------|--|
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DNs. The unit is ms. |
| cpu_skew_percent | Integer | CPU time skew of a statement among DNs. |
| min_peak_iops | Integer | Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| max_peak_iops | Integer | Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| average_peak_iops | Integer | Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| iops_skew_percent | Integer | I/O skew across DNs |
| warning | Text | Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed:
<ol style="list-style-type: none">1. The max spill size exceeds the alarm size 256MB2. The max broadcast size exceeds the alarm size 100MB3. Early spill4. Spill times is greater than 35. Spill on memory adaptive6. Hash table conflict7. NestLoop in hashjoin |
| queryid | Bigint | Internal query ID used for statement execution |
| query | Text | Statement to be executed. A maximum of 64 KB of strings can be retained. |

| Column | Type | Description |
|---------------------|--------|--|
| query_plan | Text | Execution plan of a statement
Specification restrictions:
1. Execution plans are displayed only for DML statements.
2. In 8.2.1.100 and later versions, the number of data binding times is added to the execution plans of Parse Bind Execute (PBE) statements to facilitate statement analysis. The number of data binding times is displayed in the format of PBE bind times: Times . |
| node_group | Text | Logical cluster of the user running the statement |
| pid | Bigint | PID of the backend thread for the statement. |
| lane | Text | Fast/Slow lane where the statement is executed |
| unique_sql_id | Bigint | ID of the normalized unique SQL |
| session_id | Text | Unique identifier of a session in the database system. Its format is session_start_time.tid.node_name . |
| min_read_bytes | Bigint | Minimum I/O read bytes of a statement across all DN. The unit is byte. |
| max_read_bytes | Bigint | Maximum I/O read bytes of a statement across all DN. The unit is byte. |
| average_read_bytes | Bigint | Average I/O read bytes of a statement across all DN. |
| min_write_bytes | Bigint | Minimum I/O write bytes of a statement across all DN. |
| max_write_bytes | Bigint | Maximum I/O write bytes of a statement across all DN. |
| average_write_bytes | Bigint | Average I/O write bytes of a statement across all DN. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DN. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DN. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |

| Column | Type | Description |
|-----------------------------|--------------|---|
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| stmt_type | Text | Query type corresponding to the statement. |
| except_info | Text | Information about the exception rule triggered by the statement. |
| parse_time | Bigint | Total parsing time before the statement is queued (including lexical and syntax parsing, optimization rewriting, and plan generation time), in milliseconds. This column is only supported in version 8.3.0.100 or later. |
| unique_plan_id | Bigint | ID of the normalized unique plan. |
| sql_hash | Text | Normalized SQL hash. |
| plan_hash | Text | Normalized plan hash. |
| disk_cache_hit_ratio | numeric(5,2) | Disk cache hit rate. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_disk_read_size | Bigint | Total size of data read from disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_disk_write_size | Bigint | Total size of data written to disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_remote_read_size | Bigint | Total size of data read remotely from OBS due to disk cache read failure, in MB. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_remote_read_time | Bigint | Total number of times data is read remotely from OBS due to disk cache read failure. This column only applies to OBS 3.0 tables and foreign tables. |
| vfs_scan_bytes | Bigint | Total number of bytes scanned by the OBS virtual file system in response to upper-layer requests, in bytes. This column only applies to OBS 3.0 tables and foreign tables. |
| vfs_remote_read_bytes | Bigint | Total number of bytes actually read from OBS by the OBS virtual file system, in bytes. This column only applies to OBS 3.0 tables and foreign tables. |
| preload_submit_time | Bigint | Total time for submitting I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables. |

| Column | Type | Description |
|-----------------------------|--------|---|
| preload_wait_time | Bigint | Total time for waiting for I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables. |
| preload_wait_count | Bigint | Total number of times that the prefetching process waits for I/O requests. This column only applies to OBS 3.0 tables. |
| disk_cache_load_time | Bigint | Total time for reading from disk cache, in microseconds. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_conflict_count | Bigint | Number of times a block in the disk cache produces a hash conflict. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_error_count | Bigint | Number of disk cache read failures. This column only applies to OBS 3.0 tables and foreign tables. |
| disk_cache_error_code | Bigint | Error code for disk cache read failures. This column only applies to OBS 3.0 tables and foreign tables. |
| obs_io_req_avg_rtt | Bigint | Average Round Trip Time (RTT) for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables. |
| obs_io_req_avg_latency | Bigint | Average delay for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables. |
| obs_io_req_latency_gt_1s | Bigint | Number of OBS I/O requests with a latency exceeding 1 second. This column only applies to OBS 3.0 tables and foreign tables. |
| obs_io_req_latency_gt_10s | Bigint | Number of OBS I/O requests with a latency exceeding 10 seconds. This column only applies to OBS 3.0 tables and foreign tables. |
| obs_io_req_count | Bigint | Total number of OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables. |
| obs_io_req_retry_count | Bigint | Total number of retries for OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables. |
| obs_io_req_rate_limit_count | Bigint | Total number of times OBS I/O requests are flow-controlled. This column only applies to OBS 3.0 tables and foreign tables. |

18.3.246 PGXC_WLM_SESSION_HISTORY

PGXC_WLM_SESSION_HISTORY displays load management records after job execution on all CNs. This view is used to query data from DWS, and the data in the database is cleared periodically every 3 minutes.

NOTICE

The **PGXC_WLM_SESSION_HISTORY** view can be queried only in the **postgres** database. If the view is queried in other databases, an error is reported.

GS_WLM_SESSION_HISTORY lists the columns in the **PGXC_WLM_SESSION_HISTORY** view.

Table 18-319 GS_WLM_SESSION_HISTORY columns

| Column | Type | Description |
|------------------|---------|--|
| datid | OID | OID of the database this backend is connected to |
| dbname | Text | Name of the database the backend is connected to |
| schemaname | Text | Schema name |
| nodename | Text | Name of the CN where the statement is run |
| username | Text | User name used for connecting to the backend |
| application_name | Text | Name of the application that is connected to the backend |
| client_addr | inet | IP address of the client connected to this backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number that the client uses for communication with this backend, or -1 if a Unix socket is used |
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |

| Column | Type | Description |
|---------------------|--------------------------|--|
| block_time | Bigint | Duration that a statement is blocked before being executed, including the statement parsing and optimization duration. The unit is ms. |
| start_time | Timestamp with time zone | Time when the statement starts to be run |
| finish_time | Timestamp with time zone | Time when the statement execution ends |
| duration | Bigint | Execution time of a statement. The unit is ms. |
| estimate_total_time | Bigint | Estimated execution time of a statement. The unit is ms. |
| status | Text | Final statement execution status. Its value can be finished (normal) or aborted (abnormal). The statement status here is the execution status of the database server. If the statement is successfully executed on the database server but an error is reported in the result set, the statement status is finished . |
| abort_info | Text | Exception information displayed if the final statement execution status is aborted . |
| resource_pool | Text | Resource pool used by the user |
| control_group | Text | Cgroup used by the statement |
| estimate_memory | Integer | Estimated memory used by a statement on a single instance. The unit is MB. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DN. The unit is MB. |
| average_peak_memory | Integer | Average memory usage during statement execution. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of a statement among DN. |

| Column | Type | Description |
|----------------------|---------|--|
| spill_info | Text | Statement spill information on all DNs.
None indicates that the statement has not been spilled to disks on any DNs.
All : The statement has been spilled to disks on all DNs.
[<i>a</i>:<i>b</i>] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DNs. |
| min_spill_size | Integer | Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_dn_time | Bigint | Minimum execution time of a statement across all DNs. The unit is ms. |
| max_dn_time | Bigint | Maximum execution time of a statement across all DNs. The unit is ms. |
| average_dn_time | Bigint | Average execution time of a statement across all DNs. The unit is ms. |
| dn_time_skew_percent | Integer | Execution time skew of a statement among DNs. |
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DNs. The unit is ms. |
| cpu_skew_percent | Integer | CPU time skew of a statement among DNs. |
| min_peak_iops | Integer | Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |

| Column | Type | Description |
|-------------------|---------|---|
| max_peak_iops | Integer | Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| average_peak_iops | Integer | Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| iops_skew_percent | Integer | I/O skew across DNs. |
| warning | Text | Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed:
<ol style="list-style-type: none">1. The max spill size exceeds the alarm size 256MB2. The max broadcast size exceeds the alarm size 100MB3. Early spill4. Spill times is greater than 35. Spill on memory adaptive6. Hash table conflict7. NestLoop in hashjoin |
| queryid | Bigint | Internal query ID used for statement execution |
| query | Text | Statement to be executed. A maximum of 64 KB of strings can be retained. |
| query_plan | Text | Execution plan of a statement.
Specification restrictions:
<ol style="list-style-type: none">1. Execution plans are displayed only for DML statements.2. In 8.2.1.100 and later versions, the number of data binding times is added to the execution plans of Parse Bind Execute (PBE) statements to facilitate statement analysis. The number of data binding times is displayed in the format of PBE bind times: Times. |
| node_group | Text | Logical cluster of the user running the statement |
| pid | Bigint | PID of the backend thread of the statement |
| lane | Text | Fast/Slow lane where the statement is executed |

| Column | Type | Description |
|---------------------|--------|---|
| unique_sql_id | Bigint | ID of the normalized unique SQL. |
| session_id | Text | Unique identifier of a session in the database system. Its format is session_start_time.tid.node_name . |
| min_read_bytes | Bigint | Minimum I/O read bytes of a statement across all DN. The unit is byte. |
| max_read_bytes | Bigint | Maximum I/O read bytes of a statement across all DN. The unit is byte. |
| average_read_bytes | Bigint | Average I/O read bytes of a statement across all DN. |
| min_write_bytes | Bigint | Minimum I/O write bytes of a statement across all DN. |
| max_write_bytes | Bigint | Maximum I/O write bytes of a statement across all DN. |
| average_write_bytes | Bigint | Average I/O write bytes of a statement across all DN. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DN. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DN. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| stmt_type | Text | Query type corresponding to the statement. |
| except_info | Text | Information about the exception rule triggered by the statement. |
| unique_plan_id | Bigint | ID of the normalized unique plan. |
| sql_hash | Text | Normalized SQL hash. |
| plan_hash | Text | Normalized plan hash. |
| use_plan_baseline | Text | Indicates whether the bound plan is used for executing the current statement. If it is used, the name of the plan_baseline column in pg_plan_baseline is displayed. |
| outline_name | Text | Name of the outline used for the statement plan. |

| Column | Type | Description |
|-----------------------------|--------------|---|
| loader_status | Text | The JSON string for storing import and export service information is as follows.
<ol style="list-style-type: none">1. address: indicates the IP address of the peer cluster. The port number is displayed for the source cluster.2. direction: indicates the import and export service type. The value can be gds to file, gds from file, gds to pipe, gds from pipe, copy from or copy to.3. min/max/total_lines/bytes: indicates the minimum value, maximum value, total lines, and bytes of the import and export statements on all DNs. |
| parse_time | Bigint | Total parsing time before the statement is queued (including lexical and syntax parsing, optimization rewriting, and plan generation time), in milliseconds. This column is available only in clusters of version 8.3.0.100 or later. |
| disk_cache_hit_ratio | numeric(5,2) | Disk cache hit rate. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_read_size | Bigint | Total size of data read from disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_disk_write_size | Bigint | Total size of data written to disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_size | Bigint | Total size of data read remotely from OBS due to disk cache read failure, in MB. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_remote_read_time | Bigint | Total number of times data is read remotely from OBS due to disk cache read failure. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| vfs_scan_bytes | Bigint | Total number of bytes scanned by the OBS virtual file system in response to upper-layer requests, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|--|
| vfs_remote_read_bytes | Bigint | Total number of bytes actually read from OBS by the OBS virtual file system, in bytes. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| preload_submit_time | Bigint | Total time for submitting I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_time | Bigint | Total time for waiting for I/O requests in the prefetching process, in microseconds. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| preload_wait_count | Bigint | Total number of times that the prefetching process waits for I/O requests. This column only applies to OBS 3.0 tables with storage and compute decoupled. |
| disk_cache_loaded_time | Bigint | Total time for reading from disk cache, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_conflict_count | Bigint | Number of times a block in the disk cache produces a hash conflict. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| disk_cache_error_count | Bigint | Number of disk cache read failures. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|---------------------------|--------|---|
| disk_cache_error_code | Bigint | <p>Error code for disk cache read failures. Multiple error codes may be generated. If the disk cache fails to be read, OBS remote read is initiated and cache blocks are rewritten. The error code types are as follows: This column only applies to OBS 3.0 tables and foreign tables.</p> <ul style="list-style-type: none">• 1: A hash conflict occurs in the disk cache block.• 2: The generation time of the disk cache block is later than that of the OldestXmin transaction.• 4: Invoking the pread system when reading cache files from the disk cache failed.• 8: The data version of the disk cache block does not match.• 16: The version of the data written to the write cache does not match the latest version.• 32: Opening the cache file corresponding to the cache block failed.• 64: The size of the data read from the disk cache does not match.• 128: The CSN recorded in the disk cache block does not match. |
| obs_io_req_avg_rtt | Bigint | Average Round Trip Time (RTT) for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_avg_latency | Bigint | Average delay for OBS I/O requests, in microseconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_1s | Bigint | Number of OBS I/O requests with a latency exceeding 1 second. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_latency_gt_10s | Bigint | Number of OBS I/O requests with a latency exceeding 10 seconds. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_count | Bigint | Total number of OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

| Column | Type | Description |
|-----------------------------|--------|---|
| obs_io_req_retry_count | Bigint | Total number of retries for OBS I/O requests. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |
| obs_io_req_rate_limit_count | Bigint | Total number of times OBS I/O requests are flow-controlled. This column only applies to OBS 3.0 tables and foreign tables with storage and compute decoupled. |

18.3.247 PGXC_WLM_SESSION_STATISTICS

PGXC_WLM_SESSION_STATISTICS displays load management information about jobs that are being executed on CNs.

Table 18-320 PGXC_WLM_SESSION_STATISTICS columns

| Column | Type | Description |
|------------------|---------|--|
| datid | OID | OID of the database this backend is connected to |
| dbname | Name | Name of the database the backend is connected to |
| schemaname | Text | Schema name |
| nodename | Text | Name of the CN where the statement is executed |
| username | Name | User name used for connecting to the backend |
| application_name | Text | Name of the application that is connected to the backend |
| client_addr | inet | IP address of the client connected to this backend. If this column is null, it indicates either that the client is connected via a Unix socket on the server machine or that this is an internal process such as autovacuum. |
| client_hostname | Text | Host name of the connected client, as reported by a reverse DNS lookup of client_addr . This column will only be non-null for IP connections, and only when log_hostname is enabled. |
| client_port | Integer | TCP port number used by the client to communicate with the backend. If a Unix socket is used, it is -1 . |

| Column | Type | Description |
|---------------------|--------------------------|--|
| query_band | Text | Job type, which can be set using the GUC parameter query_band and is a null string by default. |
| pid | Bigint | ID of the backend thread |
| block_time | Bigint | Block time before the statement is executed. The unit is ms. |
| start_time | Timestamp with time zone | Time when the statement starts to be executed |
| duration | Bigint | For how long a statement has been executing. The unit is ms. |
| estimate_total_time | Bigint | Estimated execution time of a statement. The unit is ms. |
| estimate_left_time | Bigint | Estimated remaining time of statement execution. The unit is ms. |
| enqueue | Text | Workload management resource status |
| resource_pool | Name | Resource pool used by the user |
| control_group | Text | Cgroup used by the statement |
| estimate_memory | Integer | Estimated memory used by a statement on a single instance. The unit is MB. |
| min_peak_memory | Integer | Minimum memory peak of a statement across all DN. The unit is MB. |
| max_peak_memory | Integer | Maximum memory peak of a statement across all DN. The unit is MB. |
| average_peak_memory | Integer | Average memory usage during statement execution. The unit is MB. |
| memory_skew_percent | Integer | Memory usage skew of a statement among DN. |
| spill_info | Text | Spill information for the statement on all DN. The options are:
None : The statement has not been spilled to disks on any DN.
All : The statement has been spilled to disks on all DN.
[a:b] : The statement has been spilled to disks on <i>a</i> of <i>b</i> DN. |

| Column | Type | Description |
|----------------------|---------|---|
| min_spill_size | Integer | Minimum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| max_spill_size | Integer | Maximum spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| average_spill_size | Integer | Average spilled data among all DNs when a spill occurs. The unit is MB. The default value is 0. |
| spill_skew_percent | Integer | DN spill skew when a spill occurs |
| min_dn_time | Bigint | Minimum execution time of a statement across all DNs. The unit is ms. |
| max_dn_time | Bigint | Maximum execution time of a statement across all DNs. The unit is ms. |
| average_dn_time | Bigint | Average execution time of a statement across all DNs. The unit is ms. |
| dn_time_skew_percent | Integer | Execution time skew of a statement among DNs. |
| min_cpu_time | Bigint | Minimum CPU time of a statement across all DNs. The unit is ms. |
| max_cpu_time | Bigint | Maximum CPU time of a statement across all DNs. The unit is ms. |
| total_cpu_time | Bigint | Total CPU time of a statement across all DNs. The unit is ms. |
| cpu_skew_percent | Integer | CPU time skew of a statement among DNs. |
| min_peak_iops | Integer | Minimum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| max_peak_iops | Integer | Maximum IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |
| average_peak_iops | Integer | Average IOPS peak of a statement across all DNs. It is counted by ones in a column-store table and by ten thousands in a row-store table. |

| Column | Type | Description |
|---------------------|---------|--|
| iops_skew_percent | Integer | I/O skew across DNs. |
| min_read_speed | Integer | Minimum I/O read rate of a statement across all DNs within a monitoring period (5s). The unit is KB/s. |
| max_read_speed | Integer | Maximum I/O read rate of a statement across all DNs within a monitoring period (5s). The unit is KB/s. |
| average_read_speed | Integer | Average I/O read rate of a statement across all DNs within a monitoring period (5s). The unit is KB/s. |
| min_write_speed | Integer | Minimum I/O write rate of a statement across all DNs within a monitoring period (5s). The unit is KB/s. |
| max_write_speed | Integer | Maximum I/O write rate of a statement across all DNs within a monitoring period (5s). The unit is KB/s. |
| average_write_speed | Integer | Average I/O write rate of a statement across all DNs within a monitoring period (5s). The unit is KB/s. |
| recv_pkg | Bigint | Total number of communication packages received by a statement across all DNs. |
| send_pkg | Bigint | Total number of communication packages sent by a statement across all DNs. |
| recv_bytes | Bigint | Total received data of the statement stream, in byte. |
| send_bytes | Bigint | Total sent data of the statement stream, in byte. |
| warning | Text | Warning. The following warnings and warnings related to SQL self-diagnosis tuning are displayed:
<ol style="list-style-type: none">1. The max spill size exceeds the alarm size 256MB2. The max broadcast size exceeds the alarm size 100MB3. Early spill4. Spill times is greater than 35. Spill on memory adaptive6. Hash table conflict7. NestLoop in hashjoin |

| Column | Type | Description |
|----------------------------|---------------|--|
| unique_sql_id | Bigint | ID of the normalized unique SQL. |
| queryid | Bigint | Internal query ID used for statement execution |
| query | Text | Statement that is being executed |
| query_plan | Text | Execution plan of a statement
Specification restrictions:
1. Execution plans are displayed only for DML statements.
2. In 8.2.1.100 and later versions, the number of data binding times is added to the execution plans of Parse Bind Execute (PBE) statements to facilitate statement analysis. The number of data binding times is displayed in the format of PBE bind times: Times . |
| node_group | Text | Logical cluster of the user running the statement |
| stmt_type | Text | Query type corresponding to the statement. |
| except_info | Text | Information about the exception rule triggered by the statement. |
| parse_time | Bigint | Total parsing time before the statement is queued (including lexical and syntax parsing, optimization rewriting, and plan generation time), in milliseconds. This column is only supported in version 8.3.0.100 or later. |
| unique_plan_id | Bigint | ID of the normalized unique plan. |
| sql_hash | Text | Normalized SQL hash. |
| plan_hash | Text | Normalized plan hash. |
| disk_cache_hit_ratio | numeric(5, 2) | Disk cache hit rate. This column only applies to OBS 3.0 tables and foreign tables in decoupled storage and compute scenarios. |
| disk_cache_disk_read_size | Bigint | Total size of data read from disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables in decoupled storage and compute scenarios. |
| disk_cache_disk_write_size | Bigint | Total size of data written to disk cache, in MB. This column only applies to OBS 3.0 tables and foreign tables in decoupled storage and compute scenarios. |

| Column | Type | Description |
|-----------------------------|--------|--|
| disk_cache_remote_read_size | Bigint | Total size of data read remotely from OBS due to disk cache read failure, in MB. This column only applies to OBS 3.0 tables and foreign tables in decoupled storage and compute scenarios. |
| disk_cache_remote_read_time | Bigint | Total number of times data is read remotely from OBS due to disk cache read failure. This column only applies to OBS 3.0 tables and foreign tables in decoupled storage and compute scenarios. |
| block_name | Text | Name of the interception rule that matches the statement. |

18.3.248 PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS

PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS displays data skews of tables in the current database. You can quickly query the storage space skew of all tables in the current database on each node. This view is supported only by clusters of version 8.2.1 or later.

Suggestions:

- When you analyze the disk space skew of each table in a database in a large cluster with a large amount of data, the **PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS** view delivers better query performance than the **gs_table_distribution()** function and the **PGXC_GET_TABLE_SKEWNESS** view. You are advised to use the **PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS** view to query the table skew status overview, and then use the **gs_table_distribution(schemaname text, tablename text)** function to obtain the disk space distribution of a specified table on each node.
- To use this view to query the storage distribution information of a specified table, you must have the **SELECT** permission on the table.
- This function is based on the physical file storage space recorded in the **PG_RELFILENODE_SIZE** system catalog. Ensure that the GUC parameters **use_workload_manager** and **enable_perm_space** are enabled.

Table 18-321 PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS columns

| Column | Type | Description |
|-------------|---------|---|
| schema_name | Name | Name of the schema where a table is |
| table_name | Name | Table name |
| total_size | Numeric | Total storage space of a table on all nodes, in bytes |

| Column | Type | Description |
|--------------|-----------------|---|
| avg_size | numeric(1000,0) | Average storage space of a table on each node, in bytes |
| max_percent | Numeric | Percentage (%) of the maximum storage space of a table on each node to the total storage space |
| min_percent | Numeric | Percentage (%) of the minimum storage space of a table on each node to the total storage space |
| skew_percent | Numeric | Skew rate (%) of a table
The formula for calculating the skew rate as follows: Skew rate (skew_percent) = (Maximum value of dnsize - Average value of dnsize) x 100/Maximum value of dnsize . |

Example

You can use the **PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS** view to query the table skew status overview, and then use the **gs_table_distribution(schemaname text, tablename text)** function to obtain the disk space distribution of a specified table on each node.

Step 1 Use the **PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS** view to query the table skew status overview.

```
tpcds_col=# select * from pgxc_wlm_table_distribution_skewness;
```

The query result is as follows:

| tpcds_col=# select * from pgxc_wlm_table_distribution_skewness; | schema_name | table_name | total_size | avg_size | max_percent | min_percent | skew_percent |
|---|-------------|-----------------------------------|------------|----------|-------------|-------------|--------------|
| | pg_catalog | pg_namespace_oid_index | 98304 | 16384 | 16.67 | 16.67 | 0.00 |
| | public | customer_demographics | 4595712 | 1531904 | 33.33 | 33.33 | 0.00 |
| | pg_catalog | pg_type_oid_index | 245760 | 40960 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_partition | 49152 | 8192 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_user_mapping_user_server_index | 49152 | 8192 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_attrdef | 294912 | 49152 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_global_temp_parent_nsp_index | 49152 | 8192 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_namespace_nspname_index | 98304 | 16384 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_synonym_name_nsp_index | 49152 | 8192 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_proc_proname_args_nsp_index | 2015232 | 335872 | 16.67 | 16.67 | 0.00 |
| | public | dbgen_version | 32768 | 10923 | 100.00 | 0.00 | 66.67 |
| | pg_catalog | pg_amop_opr_fam_index | 294912 | 49152 | 16.67 | 16.67 | 0.00 |
| | pg_catalog | pg_global_temp | 0 | 0 | 0.00 | 0.00 | 0.00 |
| | pg_catalog | pg_workload_group_name_index | 0 | 0 | 0.00 | 0.00 | 0.00 |
| | public | date_dim | 7159808 | 2386603 | 34.10 | 32.95 | 2.24 |
| | pg_catalog | pg_operator_oid_index | 245760 | 40960 | 16.67 | 16.67 | 0.00 |

The data skew of the **dbgen_version** table is severe.

Step 2 Use the **gs_table_distribution(schemaname text, tablename text)** function to query the disk space distribution of the **dbgen_version** table on each node.

```
tpcds_col=# select * from gs_table_distribution('public','dbgen_version');
```

The query result is as follows:

| tpcds_col=# select * from gs_table_distribution('public','dbgen_version'); | schemaname | tablename | relkind | relpersistence | nodename | dnsize | sessionid |
|--|------------|---------------|---------|----------------|--------------|--------|-----------|
| | public | dbgen_version | r | p | dn_6001_6002 | 0 | |
| | public | dbgen_version | r | p | dn_6005_6006 | 32768 | |
| | public | dbgen_version | r | p | dn_6003_6004 | 0 | |
| (3 rows) | | | | | | | |

According to the preceding information, data skew occurs in the disk space occupied by the table on DNs. Most data is stored on **dn_6005_6006**.

----End

18.3.249 PGXC_WLM_USER_RESOURCE_HISTORY

The **PGXC_WLM_USER_RESOURCE_HISTORY** view displays historical information about resource consumption of all users on the corresponding instances. This view is supported only by clusters of version 8.2.0 or later.

Table 18-322 PGXC_WLM_USER_RESOURCE_HISTORY columns

| Column | Type | Description |
|--------------|--------------------------|--|
| nodename | Name | Instance name, including CNs and DNs. |
| username | Text | Username |
| Timestamp | Timestamp with time zone | Timestamp |
| used_memory | Integer | Used memory (unit: MB).
On a DN, it indicates a user's memory usage on the current DN.
On a CN, it indicates a user's total memory usage on all DNs. |
| total_memory | Integer | Available memory (unit: MB). 0 indicates that the available memory is not limited and depends on the maximum memory available in the database.
On a DN, it indicates the memory available to a user on the current DN.
On a CN, it indicates the total memory available to a user on all DNs. |
| used_cpu | Double precision | Number of CPU cores in use. Only the CPU usage of complex jobs in the non-default resource pool is collected, and the value is the CPU usage of the related cgroup.
On a DN, it indicates a user's CPU core usage on the current DN.
On a CN, it indicates a user's total CPU core usage on all DNs. |

| Column | Type | Description |
|------------------|---------|--|
| total_cpu | Integer | Total number of CPU cores of the Cgroups associated with a user.
On a DN, it indicates the CPU cores available to a user on the current DN.
On a CN, it indicates the total CPU cores available to a user on all DNs. |
| used_space | Bigint | Used permanent table storage space (unit: KB)
On a DN, it indicates the size of the permanent table storage space used by a user on the current DN.
On a CN, it indicates the total size of the permanent table storage space used by a user on all DNs. |
| total_space | Bigint | Available storage space, in KB. -1 indicates that the storage space is not limited.
On a DN, it indicates the size of the permanent table storage space available to a user on the current DN.
On a CN, it indicates the total size of the permanent table storage space available to a user on all DNs. |
| used_temp_space | Bigint | Used temporary table storage space (unit: KB)
On a DN, it indicates the size of the temporary table storage space used by a user on the current DN.
On a CN, it indicates the total size of the temporary table storage space used by a user on all DNs. |
| total_temp_space | Bigint | Available temporary table storage space, in KB. -1 indicates that the storage space is not limited.
On a DN, it indicates the size of the temporary table storage space available to a user on the current DN.
On a CN, it indicates the total size of the temporary table storage space available to a user on all DNs. |

| Column | Type | Description |
|-------------------|--------|---|
| used_spill_space | Bigint | <p>Size of space used for operator spill to disk, in KB.</p> <p>On a DN, it indicates displays the size of the operator flushing space used by the user on the current DN.</p> <p>On a CN, it indicates the total space used by a user's operators spilled to disk on all DNs.</p> |
| total_spill_space | Bigint | <p>Size of space available for operator spill to disk, in KB. The value -1 indicates that the space is not limited.</p> <p>On a DN, it indicates displays the size of the operator flushing space that can be used by the user on the current DN.</p> <p>On a CN, it indicates the total space available for a user to spill operators to disk on all DNs.</p> |
| read_kbytes | Bigint | <p>On a CN, it indicates total number of bytes read by a user's complex jobs on all DNs in the last 5 seconds. The unit is KB.</p> <p>On a DN, it indicates the total number of bytes read by a user's complex jobs from the instance startup time to the current time. The unit is KB.</p> |
| write_kbytes | Bigint | <p>On a CN, it indicates total number of bytes written by a user's complex jobs on all DNs in the last 5 seconds.</p> <p>On a DN, it indicates the total number of bytes written by a user's complex jobs from the instance startup time to the current time. The unit is KB.</p> |
| read_counts | Bigint | <p>On a CN, it indicates total number of read times of a user's complex jobs on all DNs in the last 5 seconds.</p> <p>On a DN, it indicates total number of read times of a user's complex jobs from the instance startup time to the current time.</p> |
| write_counts | Bigint | <p>On a CN, it indicates total number of write times of a user's complex jobs on all DNs in the last 5 seconds.</p> <p>On a DN, it indicates total number of write times of a user's complex jobs from the instance startup time to the current time.</p> |

| Column | Type | Description |
|-------------|------------------|---|
| read_speed | Double precision | On a CN, it indicates the average read rate of a user's complex jobs on a single DN in the last 5 seconds, in KB/s.

On a DN, it indicates the average read rate of a user's complex jobs on the DN in the last 5 seconds, in KB/s. |
| write_speed | Double precision | On a CN, it indicates the average write rate of a user's complex jobs on a single DN in the last 5 seconds, in KB/s.

On a DN, it indicates the average write rate of a user's complex jobs on the DN in the last 5 seconds, in KB/s. |
| send_speed | Double precision | On a CN, it indicates the sum of the average network sending rates of a user on all DNs in a 5s monitoring period, in KB/s.

On a DN, it indicates the average network sending rate of a user on the DN in a 5s monitoring period, in KB/s. |
| recv_speed | Double precision | On a CN, it indicates the sum of the average network receiving rates of a user on all DNs in a 5s monitoring period, in KB/s.

On a DN, it indicates the average network receiving rate of a user on the DN in a 5s monitoring period, in KB/s. |

18.3.250 PGXC_WLM_WORKLOAD_RECORDS

PGXC_WLM_WORKLOAD_RECORDS displays the status of job executed by the current user on CNs. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view. This view is available only when **enable_dynamic_workload** is set to **on**.

Table 18-323 PGXC_WLM_WORKLOAD_RECORDS columns

| Column | Type | Description |
|-----------|---------|---|
| node_name | Text | Name of the CN where the job is executed. |
| thread_id | Bigint | ID of the backend thread. |
| processid | Integer | lwpid of the thread. |
| Timestamp | Bigint | Start time of statement execution. |
| username | Name | Username logged in to the backend. |
| memory | Integer | Memory required for the statement. |

| Column | Type | Description |
|---------------|---------|--|
| active_points | Integer | Number of resources consumed by the statement on the resource pool. |
| max_points | Integer | Maximum number of resources in the resource pool. |
| priority | Integer | Priority of a job. |
| resource_pool | Text | Resource pool where a job is. |
| status | Text | Job execution status. The options are:
pending
running
finished
aborted
unknown |
| control_group | Text | Cgroups used by a job. |
| enqueue | Text | Queue for the job, including:
GLOBAL : global queue.
RESPOOL : resource pool queue.
ACTIVE : not queued. |
| query | Text | Statement currently being executed. |

18.3.251 PGXC_WORKLOAD_SQL_COUNT

PGXC_WORKLOAD_SQL_COUNT displays statistics on the number of SQL statements executed in workload Cgroups on all CNs in a cluster, including the number of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements and the number of DDL, DML, and DCL statements. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 18-324 PGXC_WORKLOAD_SQL_COUNT columns

| Column | Type | Description |
|--------------|--------|-------------------------------------|
| node_name | Name | Node name. |
| workload | Name | Workload Cgroup name. |
| select_count | Bigint | Number of SELECT statements. |
| update_count | Bigint | Number of UPDATE statements. |

| Column | Type | Description |
|--------------|--------|-------------------------------------|
| insert_count | Bigint | Number of INSERT statements. |
| delete_count | Bigint | Number of DELETE statements. |
| ddl_count | Bigint | Number of DDL statements. |
| dml_count | Bigint | Number of DML statements. |
| dcl_count | Bigint | Number of DCL statements. |

18.3.252 PGXC_WORKLOAD_SQL_ELAPSE_TIME

PGXC_WORKLOAD_SQL_ELAPSE_TIME displays statistics on the response time of SQL statements in workload Cgroups on all CNs in a cluster, including the maximum, minimum, average, and total response time of **SELECT**, **UPDATE**, **INSERT**, and **DELETE** statements. The unit is microsecond. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view.

Table 18-325 PGXC_WORKLOAD_SQL_ELAPSE_TIME columns

| Column | Type | Description |
|---------------------|--------|--|
| node_name | Name | Node name. |
| workload | Name | Workload Cgroup name. |
| total_select_elapse | Bigint | Total response time of SELECT statements. |
| max_select_elapse | Bigint | Maximum response time of SELECT statements. |
| min_select_elapse | Bigint | Minimum response time of SELECT statements. |
| avg_select_elapse | Bigint | Average response time of SELECT statements. |
| total_update_elapse | Bigint | Total response time of UPDATE statements. |
| max_update_elapse | Bigint | Maximum response time of UPDATE statements. |
| min_update_elapse | Bigint | Minimum response time of UPDATE statements. |

| Column | Type | Description |
|---------------------|--------|--|
| avg_update_elapse | Bigint | Average response time of UPDATE statements. |
| total_insert_elapse | Bigint | Total response time of INSERT statements. |
| max_insert_elapse | Bigint | Maximum response time of INSERT statements. |
| min_insert_elapse | Bigint | Minimum response time of INSERT statements. |
| avg_insert_elapse | Bigint | Average response time of INSERT statements. |
| total_delete_elapse | Bigint | Total response time of DELETE statements. |
| max_delete_elapse | Bigint | Maximum response time of DELETE statements. |
| min_delete_elapse | Bigint | Minimum response time of DELETE statements. |
| avg_delete_elapse | Bigint | Average response time of DELETE statements. |

18.3.253 PGXC_WORKLOAD_TRANSACTION

PGXC_WORKLOAD_TRANSACTION provides transaction information about workload cgroups on all CNs. Only the system administrator or the preset role **gs_role_read_all_stats** can access this view. This view is valid only when the real-time resource monitoring function is enabled, that is, when [enable_resource_track](#) is **on**.

Table 18-326 PGXC_WORKLOAD_TRANSACTION columns

| Column | Type | Description |
|------------------|--------|---|
| node_name | Name | Node name. |
| workload | Name | Workload Cgroup name. |
| commit_counter | Bigint | Number of the commits. |
| rollback_counter | Bigint | Number of rollbacks. |
| resp_min | Bigint | Minimum response time, in microseconds. |
| resp_max | Bigint | Maximum response time, in microseconds. |

| Column | Type | Description |
|------------|--------|---|
| resp_avg | Bigint | Average response time, in microseconds. |
| resp_total | Bigint | Total response time, in microseconds. |

18.3.254 PLAN_TABLE

PLAN_TABLE displays the plan information collected by **EXPLAIN PLAN**. Plan information is in a session-level life cycle. After the session exits, the data will be deleted. Data is isolated between sessions and between users.

Table 18-327 PLAN_TABLE columns

| Column | Type | Description |
|--------------|----------------|---|
| statement_id | varchar2(30) | Query tag specified by a user |
| plan_id | Bigint | ID of a plan to be queried |
| id | Int | ID of each operator in a generated plan |
| operation | varchar2(30) | Operation description of an operator in a plan |
| options | varchar2(255) | Operation parameters |
| object_name | Name | Name of an operated object. It is defined by users, not the object alias used in the query. |
| object_type | varchar2(30) | Object type |
| object_owner | Name | User-defined schema to which an object belongs |
| projection | varchar2(4000) | Returned column information |

NOTE

- A valid **object_type** value consists of a relkind type defined in **PG_CLASS** (**TABLE** ordinary table, **INDEX**, **SEQUENCE**, **VIEW**, **FOREIGN TABLE**, **COMPOSITE TYPE**, or **TOASTVALUE TOAST** table) and the rtekind type used in the plan (**SUBQUERY**, **JOIN**, **FUNCTION**, **VALUES**, **CTE**, or **REMOTE_QUERY**).
- For RangeTableEntry (RTE), **object_owner** is the object description used in the plan. Non-user-defined objects do not have **object_owner**.
- Information in the **statement_id**, **object_name**, **object_owner**, and **projection** columns is stored in letter cases specified by users and information in other columns is stored in uppercase.
- **PLAN_TABLE** supports only **SELECT** and **DELETE** and does not support other DML operations.

18.3.255 PV_FILE_STAT

By collecting statistics about the data file I/Os, **PV_FILE_STAT** displays the I/O performance of the data to detect the performance problems, such as abnormal I/O operations.

Table 18-328 PV_FILE_STAT columns

| Column | Type | Description |
|-----------|--------|---|
| filenum | OID | File ID. |
| dbid | OID | Database ID. |
| spcid | OID | Tablespace ID. |
| phyrds | Bigint | Number of physical files read. |
| phywrts | Bigint | Number of physical files written. |
| phyblkrd | Bigint | Number of physical file blocks read. |
| phyblkwrt | Bigint | Number of physical file blocks written. |
| readtim | Bigint | Total duration of file reads, in microseconds. |
| writetim | Bigint | Total duration of file writes, in microseconds. |
| avgiotim | Bigint | Average duration of file reads and writes, in microseconds. |
| lstiotim | Bigint | Duration of the last file read, in microseconds. |
| miniotim | Bigint | Minimum duration of file reads and writes, in microseconds. |
| maxiowtm | Bigint | Maximum duration of file reads and writes, in microseconds. |

18.3.256 PV_INSTANCE_TIME

PV_INSTANCE_TIME collects statistics on the running time of processes and the time consumed in each execution phase, in microseconds.

PV_INSTANCE_TIME records time consumption information of the current node. The time consumption information is classified into the following types:

- **DB_TIME**: time spent by a process on multi-core parallel computing
- **CPU_TIME**: time spent by a process on the CPU core
- **EXECUTION_TIME**: time spent on the executor
- **PARSE_TIME**: time spent on parsing SQL statements
- **PLAN_TIME**: time spent on generating plans
- **REWRITE_TIME**: time spent on rewriting SQL statements

- **PL_EXECUTION_TIME**: execution time of the PL/pgSQL stored procedure
- **PL_COMPILATION_TIME**: compilation time of the PL/pgSQL stored procedure
- **NET_SEND_TIME**: time spent on the network
- **DATA_IO_TIME**: time spent on I/O

Table 18-329 PV_INSTANCE_TIME columns

| Column | Type | Description |
|-----------|---------|---------------------------|
| stat_id | Integer | Type ID. |
| stat_name | Text | Name of the runtime type. |
| value | Bigint | Runtime value. |

18.3.257 PV_MATVIEW_DETAIL

PV_MATVIEW_DETAIL displays detailed information about a materialized view. This view is supported only by clusters of version 9.1.0 200 or later.

Table 18-330 PV_MATVIEW_DETAIL columns

| Column | Type | Description |
|--------------------|-----------|---|
| matview | Text | Materialized view name |
| baserel | Text | Base table name |
| partids | oidvector | OID of a specified partition. |
| contain_entire_rel | Boolean | Whether to create a materialized view based on the entire base table |
| build_mode | Text | Build mode of the materialized view. <ul style="list-style-type: none">• 'd': indicates "deferred", which means that data is contained in the materialized view only when the view is refreshed for the first time.• 'i': indicates "immediate", which means that the latest data is included when the materialized view is created. |
| refresh_mode | Text | Refresh mode of the materialized view.
'd': stands for demand, indicating on-demand update. |
| refresh_method | Text | Refresh method of the materialized view.
'c' indicates a full refresh. |
| mapping | Text | Mapping between base table partitions and materialized view partitions |

| Column | Type | Description |
|---------------------|--------------------------|---|
| active | Boolean | Whether the materialized view needs to be refreshed |
| refresh_start_time | Timestamp with time zone | Start time of the last refresh |
| refresh_finish_time | Timestamp with time zone | End time of the last refresh |

18.3.258 PV_OS_RUN_INFO

PV_OS_RUN_INFO displays the running status of the current operating system.

Table 18-331 PV_OS_RUN_INFO columns

| Column | Type | Description |
|------------|---------|---|
| id | Integer | ID. |
| Name | Text | Name of the operating system status. |
| value | Numeric | Value of the operating system status. |
| comments | Text | Comments on the operating system status. |
| cumulative | boolean | Whether the value of the operating system status is cumulative. |

18.3.259 PV_REDO_STAT

PV_REDO_STAT displays statistics on redoing Xlogs on the current node.

Table 18-332 PV_REDO_STAT columns

| Name | Type | Description |
|-----------|--------|------------------------------------|
| phywrts | Bigint | Number of physical writes. |
| phyblkwrt | Bigint | Number of physical blocks written. |
| writetim | Bigint | Time taken for physical writes. |
| avgiotim | Bigint | Average time taken per write. |
| lstiotim | Bigint | Time taken for the last write. |
| miniotim | Bigint | Minimum time taken for a write. |

| Name | Type | Description |
|----------|--------|---------------------------------|
| maxiowtm | Bigint | Maximum time taken for a write. |

18.3.260 PV_RUNTIME_ATTSTATS

PV_RUNTIME_ATTSTATS displays table-level statistics in the memory generated by autoanalyze. The descriptions of the columns in **PV_RUNTIME_RELSTATS** are the same as those in **PG_STATS**. This view is used only by clusters of version 8.2.0 or later.

Table 18-333 PV_RUNTIME_ATTSTATS columns

| Column | Type | Reference | Description |
|------------|---------|--|---|
| schemaname | Name | PG_NAMESP
ACE .nspname | Name of the schema that contains the table |
| tablename | Name | PG_CLASS .rel
name | Table name |
| attname | Name | PG_ATTRIBU
TE .attname | Column name |
| inherited | boolean | - | If the value is true , the inherited subcolumns are included. If the value is false , only the columns in a specified table are included. |
| null_frac | Real | - | Percentage of column entries that are null |
| avg_width | Integer | - | Average width in bytes of column's entries |

| Column | Type | Reference | Description |
|-----------------------|----------|-----------|---|
| n_distinct | Real | - | <ul style="list-style-type: none">• If the value is greater than 0, it indicates the estimated number of distinct values in the column.• Negative of the number of distinct values divided by the number of rows if the value is less than 0 <p>The negated form is used when ANALYZE believes that the number of distinct values is likely to increase as the table grows.</p> <p>The positive form is used when the column seems to have a fixed number of possible values. For example, -1 indicates a unique column in which the number of distinct values is the same as the number of rows.</p> |
| n_dndistinct | Real | - | <p>Number of unique non-null data values in the dn1 column</p> <ul style="list-style-type: none">• Exact number of distinct values if the value is greater than 0• Negative of the number of distinct values divided by the number of rows if the value is less than 0 (For example, if the value of a column appears twice in average, set n_dndistinct=-0.5.)• The number of distinct values is unknown if the value is 0. |
| most_commo
n_vals | anyarray | - | List of the most common values in a column. If this combination does not have the most common values, it will be NULL . |
| most_commo
n_freqs | real[] | - | List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals is NULL) |

| Column | Type | Reference | Description |
|------------------------|----------|-----------|---|
| histogram_bounds | anyarray | - | List of values that divide the column's values into groups of equal proportion. The values in most_common_vals , if present, are omitted from this histogram calculation. This field is null if the field data type does not have a < operator or if the most_common_vals list accounts for the entire population. |
| correlation | Real | - | Statistical correlation between physical row ordering and logical ordering of the column values. It ranges from -1 to +1. When the value is near to -1 or +1, an index scan on the column is estimated to be cheaper than when it is near to zero, due to reduction of random access to the disk. This column is null if the column data type does not have a < operator. |
| most_common_elems | anyarray | - | A list of the most commonly used non-null element values |
| most_common_elem_freqs | real[] | - | A list of the frequencies of the most commonly used element values |
| elem_count_histogram | real[] | - | A histogram of the counts of distinct non-null element values |
| partname | Name | - | Partition name This column is available only in clusters of version 8.2.1.200 or later. |

18.3.261 PV_RUNTIME_EXPRSTATS

PV_RUNTIME_EXPRSTATS displays expression statistics generated by **autoanalyze** in the memory. This view is supported only by clusters of version 8.2.1.200 or later.

Table 18-334 PV_RUNTIME_EXPRSTATS columns

| Column | Type | Reference | Description |
|------------|------|---|--|
| schemaname | Name | PG_NAMESP
ACE.nspname | Name of the schema that contains the table |

| Column | Type | Reference | Description |
|---------------------------|---------|---------------------------------------|---|
| tablename | Name | PG_CLASS .rel
name | Table name |
| statistics_sche
maname | Name | - | Name of the schema where
expression objects whose statistics
are collected is located |
| statistics_nam
e | Name | - | Name of the expression object
whose statistics are collected |
| statistics_own
er | Name | - | Owner of the expression object
whose statistics are collected |
| expr | Text | - | Expression |
| inherited | boolean | - | If the value is true , the inherited
subcolumns are included. If the
value is false , only the columns in
a specified table are included. |
| null_frac | Real | - | Percentage of column entries that
are null |
| avg_width | Integer | - | Average width in bytes of column's
entries |
| n_distinct | Real | - | <ul style="list-style-type: none">• If the value is greater than 0, it
indicates the estimated number
of distinct values in the column.• Negative of the number of
distinct values divided by the
number of rows if the value is
less than 0 <p>The negated form is used when
ANALYZE believes that the number
of distinct values is likely to
increase as the table grows.</p> <p>The positive form is used when the
column seems to have a fixed
number of possible values. For
example, -1 indicates a unique
column in which the number of
distinct values is the same as the
number of rows.</p> |

| Column | Type | Reference | Description |
|-------------------|----------|-----------|--|
| n_dndistinct | Real | - | Number of unique non-null data values in the dn1 column <ul style="list-style-type: none">Exact number of distinct values if the value is greater than 0Negative of the number of distinct values divided by the number of rows if the value is less than 0 (For example, if the value of a column appears twice in average, set n_dndistinct=-0.5.)The number of distinct values is unknown if the value is 0. |
| most_common_vals | anyarray | - | List of the most common values in a column. If this combination does not have the most common values, it will be NULL . |
| most_common_freqs | real[] | - | List of the frequencies of the most common values, that is, the number of occurrences of each value divided by the total number of rows. (NULL if most_common_vals is NULL) |
| histogram_bounds | anyarray | - | List of values that divide the column's values into groups of equal proportion. The values in most_common_vals , if present, are omitted from this histogram calculation. This field is null if the field data type does not have a < operator or if the most_common_vals list accounts for the entire population. |
| correlation | Real | - | Statistical correlation between physical row ordering and logical ordering of the column values. It ranges from -1 to +1. When the value is near to -1 or +1, an index scan on the column is estimated to be cheaper than when it is near to zero, due to reduction of random access to the disk. This column is null if the column data type does not have a < operator. |
| most_common_elems | anyarray | - | A list of the most commonly used non-null element values |

| Column | Type | Reference | Description |
|------------------------|--------|-----------|--|
| most_common_elem_freqs | real[] | - | A list of the frequencies of the most commonly used element values |
| elem_count_histogram | real[] | - | A histogram of the counts of distinct non-null element values |
| level | Text | - | Current phase of the memory statistics generated by the lightweight autoanalyze. It can be local , sendlist , or global . |

18.3.262 PV_RUNTIME_RELSTATS

PV_RUNTIME_RELSTATS displays table-level statistics in the memory generated by autoanalyze. The descriptions of the columns in **PV_RUNTIME_RELSTATS** are the same as those in **PG_CLASS**. This view is used only by clusters of version 8.2.0 or later.

Table 18-335 PV_RUNTIME_RELSTATS columns

| Name | Type | Description |
|---------------|------------------|--|
| nspname | Name | Schema name. |
| relname | Name | Name of an object, such as a table or index. |
| relpages | Double precision | Size of the on-disk representation of this table in pages (of size BLCKSZ). This is only an estimate used by the optimizer. |
| reltuples | Double precision | Number of rows in the table. This is only an estimate used by the optimizer. |
| relallvisible | Integer | Number of pages marked as all visible in the table. This column is used by the optimizer for optimizing SQL execution. |
| relhasindex | Boolean | Its value is true if this column is a table and has (or recently had) at least one index. It is set by CREATE INDEX but is not immediately cleared by DROP INDEX . If the VACUUM process detects that a table has no index, it clears the relhasindex column and sets the value to false . |
| changes | Bigint | Total historical modifications in the table by the time the lightweight autoanalyze is triggered. |

| Name | Type | Description |
|----------|------|--|
| level | Text | Current phase of the memory statistics generated by the lightweight autoanalyze. It can be local , sendlist , or global . |
| partname | Name | Partition name This column is available only in clusters of version 8.2.1.200 or later. |

18.3.263 PV_SESSION_MEMORY

PV_SESSION_MEMORY displays statistics about memory usage at the session level in the unit of MB, including all the memory allocated to Postgres and Stream threads on DNs for jobs currently executed by users.

Table 18-336 PV_SESSION_MEMORY columns

| Column | Type | Description |
|----------|---------|---|
| sessid | Text | Thread start time + Thread ID + Node name (string format: <i>timestamp.threadid.nodename</i>)
NOTE
In cluster versions earlier than 8.2.1, sessid contains only the thread start time and thread ID. In cluster versions 8.2.1 and later, the node name is added to sessid , which is in the following format: <i>timestamp.threadid.nodename</i> . |
| init_mem | Integer | Memory allocated to the currently executed task before the task enters the executor, in MB. |
| used_mem | Integer | Memory allocated to the currently executed task, in MB. |
| peak_mem | Integer | Peak memory allocated to the currently executed task, in MB. |

18.3.264 PV_SESSION_MEMORY_DETAIL

PV_SESSION_MEMORY_DETAIL displays statistics about thread memory usage by memory context.

The memory context TempSmallContextGroup collects information about all memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the number of the collected memory contexts is recorded in the **usedsize** column. Therefore, the **totalsize** and **freesize** columns for TempSmallContextGroup in the view display the corresponding information about all the memory contexts whose value in the **totalsize** column is less than 8192 bytes in the current thread, and the **usedsize** column displays the number of these memory contexts.

You can run the **SELECT * FROM pv_session_memctx_detail (threadid, '');** statement to record information about all memory contexts of a thread into the

threadid_timestamp.log file in the */tmp/dumpmem* directory. *threadid* can be obtained from the following table.

Table 18-337 PV_SESSION_MEMORY_DETAIL columns

| Column | Type | Description |
|-------------|----------|---|
| sessid | Text | Thread start time + Thread ID + Node name (string format: <i>timestamp.threadid.nodename</i>) |
| sesstype | Text | Thread name |
| contextname | Text | Name of the memory context |
| level | Smallint | Hierarchy of the memory context |
| parent | Text | Name of the parent memory context |
| totalsize | Bigint | Total size of the memory context, in bytes |
| freesize | Bigint | Total size of released memory in the memory context, in bytes |
| usedsize | Bigint | Size of used memory in the memory context, in bytes. For TempSmallContextGroup, this parameter specifies the number of collected memory contexts. |

Example

Query the usage of all MemoryContexts on the current node.

Locate the thread in which the MemoryContext is created and used based on **sessid**. Check whether the memory usage meets the expectation based on **totalsize**, **freesize**, and **usedsize** to see whether memory leakage may occur.

```
SELECT * FROM pv_session_memory_detail;
      sessid      |      sesstype      |      contextname      | level |
parent      | totalsize | freesize | usedsize
-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector |
TempSmallContextGroup | 0 | 646520 | 125184 | 149
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector |
TopMemoryContext | 0 | 372672 | 19488 | 353184
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector | Record information
cache | 1 | TopMemoryContext | 24576 | 13632 | 10944
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector | Operator class
cache | 1 | TopMemoryContext | 24576 | 15168 | 9408
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector | smgr relation
table | 1 | TopMemoryContext | 24576 | 11040 | 13536
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector |
TransactionAbortContext | 1 | TopMemoryContext | 32768 | 32704 | 64
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector | Portal
hash | 1 | TopMemoryContext | 24576 | 16192 | 8384
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector |
CacheMemoryContext | 1 | TopMemoryContext | 1858848 | 10640 |
1848208
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector |
```

```
InvalidCacheMemoryContext | 2 | CacheMemoryContext | 2088960 | 828928 |  
1260032  
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector | WAL record  
construction | 1 | TopMemoryContext | 49792 | 6528 | 43264  
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector | LOCAL LOCK  
hash | 1 | TopMemoryContext | 24576 | 15936 | 8640  
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector |  
Timezones | 1 | TopMemoryContext | 24576 | 15808 | 8768  
1760444600.140326303563096.coordinator1 | GlobalDeadLockDetector | SELF MEMORY  
CONTEXT | 1 | TopMemoryContext | 66600 | 17824 | 48776  
1760444600.140326303562480.coordinator1 | file_recyclebin |  
TempSmallContextGroup | 0 | | 642688 | 129040 | 147  
1760444600.140326303562480.coordinator1 | file_recyclebin | TopMemoryContext  
| 0 | | 1437632 | 14256 | 1423376  
1760444600.140326303562480.coordinator1 | file_recyclebin | Operator class cache  
| 1 | TopMemoryContext | 24576 | 15168 | 9408  
1760444600.140326303562480.coordinator1 | file_recyclebin | smgr relation table  
| 1 | TopMemoryContext | 24576 | 11040 | 13536  
1760444600.140326303562480.coordinator1 | file_recyclebin | TransactionAbortContext  
| 1 | TopMemoryContext | 32768 | 32704 | 64  
1760444600.140326303562480.coordinator1 | file_recyclebin | Portal hash |  
1 | TopMemoryContext | 24576 | 16192 | 8384  
1760444600.140326303562480.coordinator1 | file_recyclebin |  
CacheMemoryContext | 1 | TopMemoryContext | 1858848 | 12096 |  
1846752
```

18.3.265 PV_SESSION_STAT

PV_SESSION_STAT displays session state statistics based on session threads or the **AutoVacuum** thread.

Table 18-338 PV_SESSION_STAT columns

| Column | Type | Description |
|----------|---------|--|
| sessid | Text | Thread start time + Thread ID + Node name (string format: <i>timestamp.threadid.nodename</i>) |
| statid | Integer | Statistics ID. |
| statname | Text | Name of the statistics session. |
| statunit | Text | Unit of the statistics session. |
| value | Bigint | Value of the statistics session. |

18.3.266 PV_SESSION_TIME

PV_SESSION_TIME displays statistics about the running time of session threads and time consumed in each execution phase, in microseconds.

Table 18-339 PV_SESSION_TIME columns

| Column | Type | Description |
|--------|------|--|
| sessid | Text | Thread start time + Thread ID + Node name (string format: <i>timestamp.threadid.nodename</i>) |

| Column | Type | Description |
|-----------|---------|---------------------------|
| stat_id | Integer | Statistics ID. |
| stat_name | Text | Name of the runtime type. |
| value | Bigint | Runtime value. |

18.3.267 PV_TOTAL_MEMORY_DETAIL

PV_TOTAL_MEMORY_DETAIL displays statistics about memory usage of the current database node in the unit of MB.

Table 18-340 PV_TOTAL_MEMORY_DETAIL columns

| Name | Type | Description |
|----------|------|-------------|
| nodename | Text | Node name |

| Name | Type | Description |
|------------|------|---|
| memorytype | Text | <p>Memory type. Its value can be:</p> <ul style="list-style-type: none">• max_process_memory: memory used by a DWS cluster instance• process_used_memory: memory used by a DWS process• max_dynamic_memory: maximum dynamic memory• dynamic_used_memory: used dynamic memory• dynamic_peak_memory: dynamic peak value of the memory• dynamic_used_shrctx: maximum dynamic shared memory context• dynamic_peak_shrctx: dynamic peak value of the shared memory context• max_shared_memory: maximum shared memory• shared_used_memory: used shared memory• max_cstore_memory: maximum memory allowed for column store• cstore_used_memory: memory used for column store• max_sctpcomm_memory: maximum memory allowed for the communication library• sctpcomm_used_memory: memory used for the communication library• sctpcomm_peak_memory: memory peak of the communication library• other_used_memory: other used memory• gpu_max_dynamic_memory: maximum GPU memory• gpu_dynamic_used_memory: sum of the available GPU memory and temporary GPU memory• gpu_dynamic_peak_memory: maximum memory used for GPU• pooler_conn_memory: memory used for pooler connections• pooler_freeconn_memory: memory used for idle pooler connections• storage_compress_memory: memory used for column-store compression and decompression• udf_reserved_memory: memory reserved for the UDF Worker process |

| Name | Type | Description |
|-------------|---------|--|
| | | <ul style="list-style-type: none">• mmap_used_memory: memory used for mmap |
| memorybytes | Integer | Size of allocated memory-typed memory |

18.3.268 REDACTION_COLUMNS

REDACTION_COLUMNS displays information about all redaction columns in the current database.

Table 18-341 REDACTION_COLUMNS columns

| Name | Type | Description |
|------------------------|---------|--|
| object_schema | name | Redacted object schema. |
| object_owner | Name | Redacted object owner. |
| object_name | Name | Redacted object name. |
| column_name | Name | Redacted column name. |
| function_type | Integer | Redaction type. |
| function_parameters | Text | Parameter used when the redaction type is partial (reserved). |
| regexp_pattern | Text | Pattern string when the redaction type is regexp (reserved). |
| regexp_replace_string | Text | Replacement string when the redaction type is regexp (reserved). |
| regexp_position | Integer | Start and end replacement positions when the redaction type is regexp (reserved). |
| regexp_occurrence | Integer | Replacement times when the redaction type is regexp (reserved). |
| regexp_match_parameter | Text | Regular control parameter used when the redaction type is regexp (reserved). |
| function_info | Text | Redaction function information. |

| Name | Type | Description |
|--------------------|------|--|
| column_description | Text | Description of the redacted column. |
| inherited | Bool | Whether a redacted column is inherited from another redacted column. |
| policy_name | Name | Name of the data masking policy. This parameter is supported only by clusters of version 8.2.1.100 or later. |

18.3.269 REDACTION_POLICIES

REDACTION_POLICIES displays information about all redaction objects in the current database.

Table 18-342 REDACTION_POLICIES columns

| Name | Type | Description |
|--------------------|---------|--|
| object_schema | name | Redacted object schema. |
| object_owner | Name | Redacted object owner. |
| object_name | Name | Redacted object name. |
| policy_name | Name | Name of the redaction policy. |
| expression | Text | Policy effective expression (for users). |
| enable | Boolean | Policy status (enabled or disabled). |
| policy_description | Text | Policy description. |
| inherited | Bool | Whether a redacted column is inherited from another redacted column. |

18.3.270 REMOTE_TABLE_STAT

REMOTE_TABLE_STAT provides statistics of all tables of the database on all DNs in the cluster. Except the **nodename** column of the name type added in front of

each row, the names, types, and sequences of other columns are the same as those in the [GS_TABLE_STAT](#) view.

Table 18-343 REMOTE_TABLE_STAT columns

| Name | Type | Description |
|-------------------|--------|---|
| nodename | Name | Node name |
| schemaname | Name | Table namespace |
| relname | Name | Table name |
| seq_scan | Bigint | Number of sequential scans. Only row-store tables are counted. For a partitioned table, the sum of the number of scans of each partition is displayed. |
| seq_tuple_read | Bigint | Number of rows scanned in sequence. Only row-store tables are counted. |
| index_scan | Bigint | Number of index scans. Only row-store tables are counted. |
| index_tuple_read | Bigint | Number of rows scanned by the index. Only row-store tables are counted. |
| tuple_inserted | Bigint | Number of rows inserted |
| tuple_updated | Bigint | Number of rows updated |
| tuple_deleted | Bigint | Number of rows deleted |
| tuple_hot_updated | Bigint | Number of rows with HOT updates. |
| live_tuples | Bigint | Number of live tuples. Query the view on the CN. If ANALYZE is executed, the total number of live tuples in the table is displayed. Otherwise, 0 is displayed. This indicator applies only to row-store tables. |
| dead_tuples | Bigint | Number of dead tuples. Query the view on the CN. If ANALYZE is executed, the total number of dead tuples in the table is displayed. Otherwise, 0 is displayed. This indicator applies only to row-store tables. |

18.3.271 SHOW_TSC_INFO

Queries TSC information about the current node. This view is supported only by clusters of version 8.2.1 or later.

Table 18-344 Parameter

| Name | Type | Description |
|-----------------------|---------|---|
| node_name | text | Node name |
| tsc_mult | bigint | TSC conversion multiplier |
| tsc_shift | bigint | TSC conversion shifts |
| tsc_frequency | float8 | TSC frequency. |
| tsc_use_frequency | boolean | Indicates whether to use the TSC frequency for time conversion. |
| tsc_ready | boolean | Indicates whether the TSC frequency can be used for time conversion |
| tsc_scalar_error_info | text | Error information about obtaining TSC conversion information |
| tsc_freq_error_info | text | Error information about obtaining TSC frequency information |

18.3.272 SHOW_ALL_TSC_INFO

Queries TSC information about all nodes. This view is supported only by clusters of version 8.2.1 or later.

Table 18-345 Parameter

| Name | Type | Description |
|-----------------------|---------|---|
| node_name | text | Node name |
| tsc_mult | bigint | TSC conversion multiplier |
| tsc_shift | bigint | TSC conversion shifts |
| tsc_frequency | float8 | TSC frequency. |
| tsc_use_frequency | boolean | Indicates whether to use the TSC frequency for time conversion. |
| tsc_ready | boolean | Indicates whether the TSC frequency can be used for time conversion |
| tsc_scalar_error_info | text | Error information about obtaining TSC conversion information |
| tsc_freq_error_info | text | Error information about obtaining TSC frequency information |

18.3.273 USER_COL_COMMENTS

USER_COL_COMMENTS stores the column comments of the tables and views that the current user can access.

| Column | Type | Description |
|-------------|-----------------------|----------------------------|
| column_name | character varying(64) | Column name |
| table_name | character varying(64) | Table or view name |
| owner | character varying(64) | Owner of the table or view |
| comments | Text | Comments |

18.3.274 USER_CONSTRAINTS

USER_CONSTRAINTS displays the table constraint information accessible to the current user.

| Column | Type | Description |
|-----------------|------------------------|---|
| constraint_name | vcharacter varying(64) | Constraint name |
| constraint_type | Text | Constraint type <ul style="list-style-type: none">• C: Check constraint• F: Foreign key constraint• P: Primary key constraint• U: Unique constraint. |
| table_name | character varying(64) | Name of constraint-related table |
| index_owner | character varying(64) | Owner of constraint-related index (only for the unique constraint and primary key constraint) |
| index_name | character varying(64) | Name of constraint-related index (only for the unique constraint and primary key constraint) |

Example

Query constraints on a specified table of the current user. Replace **t1** with the actual table name.

```
SELECT * FROM USER_CONSTRAINTS WHERE table_name='t1';
constraint_name | constraint_type | table_name | index_owner | index_name
-----+-----+-----+-----+-----
c_custkey_key   | p               | t1         | u1          | c_custkey_key
(1 row)
```

18.3.275 USER_CONS_COLUMNS

USER_CONS_COLUMNS displays the information about constraint columns in the tables accessible to the current user.

| Column | Type | Description |
|-----------------|-----------------------|-------------------------------------|
| table_name | character varying(64) | Name of constraint-related table |
| column_name | character varying(64) | Name of constraint-related column |
| constraint_name | character varying(64) | Constraint name |
| position | Smallint | Position of the column in the table |

18.3.276 USER_INDEXES

USER_INDEXES displays index information in the current schema.

| Column | Type | Description |
|-------------|-----------------------|---|
| owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_name | character varying(64) | Name of the table corresponding to the index |
| uniqueness | Text | Whether the index is unique |
| generated | character varying(1) | Whether the index name is generated by the system |
| partitioned | character(3) | Whether the index has the property of the partition table |

18.3.277 USER_IND_COLUMNS

USER_IND_COLUMNS displays column information about all indexes accessible to the current user.

| Column | Type | Description |
|-------------|-----------------------|-------------|
| index_owner | character varying(64) | Index owner |

| Column | Type | Description |
|-----------------|-----------------------|-----------------------------------|
| index_name | character varying(64) | Index name |
| table_owner | character varying(64) | Table owner |
| table_name | character varying(64) | Table name |
| column_name | Name | Column name |
| column_position | Smallint | Position of a column in the index |

18.3.278 USER_IND_EXPRESSIONS

USER_IND_EXPRESSIONS displays information about the function-based expression index accessible to the current user.

| Column | Type | Description |
|-------------------|-----------------------|---|
| index_owner | character varying(64) | Index owner |
| index_name | character varying(64) | Index name |
| table_owner | character varying(64) | Table owner |
| table_name | character varying(64) | Table name |
| column_expression | Text | Function-based index expression of a specified column |
| column_position | Smallint | Position of a column in the index |

18.3.279 USER_IND_PARTITIONS

USER_IND_PARTITIONS displays information about index partitions accessible to the current user.

| Column | Type | Description |
|-------------|-----------------------|---|
| index_owner | character varying(64) | Name of the owner of the partitioned table index to which the index partition belongs |
| schema | character varying(64) | Schema of the partitioned index to which the index partition belongs |
| index_name | character varying(64) | Index name of the partitioned table to which the index partition belongs |

| Column | Type | Description |
|------------------------|-----------------------|--|
| partition_name | character varying(64) | Name of the index partition |
| index_partition_usable | boolean | Whether the index partition is available |
| high_value | Text | Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set.

Reserved field for forward compatibility. The parameter pretty_high_value is added in version 8.1.3 to record the information. |
| pretty_high_value | Text | Boundary of the table partition corresponding to the index partition. For a range partition, the boundary is the upper boundary. For a list partition, the boundary is the boundary value set.

The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of high_value . The output information can be collation and column data type. |
| def_tablespace_name | Name | Tablespace name of the index partition |

18.3.280 USER_JOBS

USER_JOBS displays all scheduled jobs owned by the current user. This view is accessible only to users with system administrator rights.

Table 18-346 USER_JOBS columns

| Column | Type | Description |
|-----------|---------------|--------------------------------------|
| job | INT4 | Job ID |
| log_user | name not null | User name of the job creator |
| priv_user | name not null | User name of the job executor |
| dbname | name not null | Database in which the job is created |

| Column | Type | Description |
|------------|-----------------------------|---|
| start_date | Timestamp without time zone | Job start time |
| start_suc | Text | Start time of the successful job execution |
| last_date | Timestamp without time zone | Start time of the last job execution |
| last_suc | Text | Start time of the last successful job execution |
| this_date | Timestamp without time zone | Start time of the ongoing job execution |
| this suc | Text | Same as THIS_DATE |
| next_date | Timestamp without time zone | Schedule time of the next job execution |
| next suc | Text | Same as next_date |
| broken | Text | Task status
Y : the system does not try to execute the task.
N : the system attempts to execute the task. |
| status | Char | Status of the current job. The value range is 'r', 's', 'f', 'd'. The default value is 's'. The indications are as follows: <ul style="list-style-type: none">• r: running• s: finished• f: failed• d: aborted |
| interval | Text | Time expression used to calculate the next execution time. If this parameter is set to null , the job will be executed once only. |
| failures | Smallint | Number of consecutive failures. |
| what | Text | Body of the PL/SQL blocks or anonymous block that the job executes |

18.3.281 USER_OBJECTS

USER_OBJECTS displays all database objects accessible to the current user.

| Column | Type | Description |
|---------------|--------------------------|--|
| owner | Name | Owner of the object |
| object_name | Name | Object name |
| object_id | OID | OID of the object |
| object_type | Name | Type of the object |
| namespace | OID | Namespace containing the object |
| created | Timestamp with time zone | Object creation time |
| last_ddl_time | Timestamp with time zone | Last time when the object was modified |

NOTICE

For details about the value ranges of **last_ddl_time** and **last_ddl_time**, see [PG_OBJECT](#).

18.3.282 USER_PART_INDEXES

USER_PART_INDEXES displays information about partitioned table indexes accessible to the current user.

| Column | Type | Description |
|-------------------|-----------------------|---|
| index_owner | character varying(64) | Name of the owner of the partitioned table index |
| schema | character varying(64) | Schema of the partitioned table index |
| index_name | character varying(64) | Name of the partitioned table index |
| table_name | character varying(64) | Name of the partitioned table to which the partitioned table index belongs |
| partitioning_type | Text | Partition policy of the partitioned table
NOTE
Currently, only range partitioning and list partitioning are supported. |
| partition_count | Bigint | Number of index partitions of the partitioned table index |

| Column | Type | Description |
|------------------------|---------|---|
| def_tablespace_name | Name | Tablespace name of the partitioned table index |
| partitioning_key_count | Integer | Number of partition keys of the partitioned table |

18.3.283 USER_PART_TABLES

USER_PART_TABLES displays information about partitioned tables accessible to the current user.

| Column | Type | Description |
|------------------------|-----------------------|---|
| table_owner | character varying(64) | Name of the owner of the partitioned table |
| schema | character varying(64) | Schema of the partitioned table |
| table_name | character varying(64) | Name of the partitioned table |
| partitioning_type | Text | Partition policy of the partitioned table
NOTE
Currently, only range partitioning and list partitioning are supported. |
| partition_count | Bigint | Number of partitions of the partitioned table |
| def_tablespace_name | Name | Tablespace name of the partitioned table |
| partitioning_key_count | Integer | Number of partition keys of the partitioned table |

18.3.284 USER_PROCEDURES

USER_PROCEDURES displays information about all stored procedures and functions in the current schema.

| Column | Type | Description |
|--------|-----------------------|---|
| owner | character varying(64) | Owner of the stored procedure or the function |

| Column | Type | Description |
|-----------------|-----------------------|--|
| object_name | character varying(64) | Name of the stored procedure or the function |
| argument_number | Smallint | Number of the input parameters in the stored procedure |

18.3.285 USER_SEQUENCES

USER_SEQUENCES displays sequence information in the current schema.

| Column | Type | Description |
|----------------|-----------------------|-----------------------|
| sequence_owner | character varying(64) | Owner of the sequence |
| sequence_name | character varying(64) | Name of the sequence |

18.3.286 USER_SOURCE

USER_SOURCE displays information about stored procedures or functions in this mode, and provides the columns defined by the stored procedures or the functions.

| Column | Type | Description |
|--------|-----------------------|--|
| owner | character varying(64) | Owner of the stored procedure or the function |
| Name | character varying(64) | Name of the stored procedure or the function |
| Text | Text | Definition of the stored procedure or the function |

18.3.287 USER_SYNONYMS

USER_SYNONYMS displays synonyms accessible to the current user.

Table 18-347 USER_SYNONYMS columns

| Name | Type | Description |
|--------------|------|--|
| schema_name | Text | Name of the schema the synonym belongs to. |
| synonym_name | Text | Synonym name. |

| Name | Type | Description |
|-------------------|------|--|
| table_owner | Text | Owner of the associated object. |
| table_schema_name | Text | Name of the schema the associated object belongs to. |
| table_name | Text | Name of the associated object. |

18.3.288 USER_TAB_COLUMNS

USER_TAB_COLUMNS stores information about columns of the tables and views that the current user can access.

| Column | Type | Description |
|----------------|------------------------|---|
| owner | character varying(64) | Owner of a table/view |
| table_name | character varying(64) | Table/View name |
| column_name | character varying(64) | Column name |
| data_type | character varying(128) | Data type of the column |
| column_id | Integer | Sequence number of the column when a table/view is created |
| data_length | Integer | Length of the column, in bytes |
| comments | Text | Comments |
| avg_col_len | Numeric | Average length of a column, in bytes |
| nullable | bpchar | Whether the column can be empty. For the primary key constraint and non-null constraint, the value is n. |
| data_precision | Integer | Precision of the data type. This parameter is valid for the numeric data type and NULL for other data types. |
| data_scale | Integer | Number of decimal places. This parameter is valid for the numeric data type and 0 for other data types. |

| Column | Type | Description |
|-------------|-----------------------|---|
| char_length | Numeric | Length of a column, in characters. This parameter is valid only for the varchar, nvarchar2, bpchar, and char types. |
| schema | character varying(64) | Namespace that contains the table or view. |
| kind | Text | Type of the current record. If the column belongs to a table, the value of this column is table . If the column belongs to a view, the value of this column is view . |

18.3.289 USER_TAB_COMMENTS

USER_TAB_COMMENTS displays comments about all tables and views accessible to the current user.

| Column | Type | Description |
|------------|-----------------------|----------------------------|
| owner | character varying(64) | Owner of the table or view |
| table_name | character varying(64) | Name of the table or view |
| comments | Text | Comments |

18.3.290 USER_TAB_PARTITIONS

USER_TAB_PARTITIONS displays all table partitions accessible to the current user. Each partition of a partitioned table accessible to the current user has a piece of record in **USER_TAB_PARTITIONS**.

| Column | Type | Description |
|----------------|-----------------------|--|
| table_owner | character varying(64) | Owner of the table that contains the partition |
| schema | character varying(64) | Schema of the partitioned table |
| table_name | character varying(64) | Table name |
| partition_name | character varying(64) | Name of the partition |

| Column | Type | Description |
|-------------------|------|---|
| high_value | Text | Upper boundary of a range partition or boundary value set of a list partition

Reserved field for forward compatibility. The parameter pretty_high_value is added in version 8.1.3 to record the information. |
| pretty_high_value | Text | Upper boundary of a range partition or boundary value set of a list partition

The query result is the instant decompilation output of the partition boundary expression. The output of this column is more detailed than that of high_value . The output information can be collation and column data type. |
| tablespace_name | Name | Name of the tablespace that contains the partition |

18.3.291 USER_TABLES

USER_TABLES displays table information in the current schema.

| Column | Type | Description |
|-----------------|-----------------------|--|
| owner | character varying(64) | Table owner |
| table_name | character varying(64) | Table name |
| tablespace_name | character varying(64) | Name of the tablespace that contains the table |
| status | character varying(8) | Whether the current record is valid |
| temporary | character(1) | Whether the table is a temporary table <ul style="list-style-type: none">• Y indicates that it is a temporary table.• N indicates that it is not a temporary table. |

| Column | Type | Description |
|----------|-------------------|--|
| dropped | character varying | Whether the current record is deleted <ul style="list-style-type: none">• YES indicates that it is deleted.• NO indicates that it is not deleted. |
| num_rows | Numeric | Estimated number of rows in the table |

18.3.292 USER_TRIGGERS

USER_TRIGGERS displays the information about triggers accessible to the current user.

| Column | Type | Description |
|--------------|-----------------------|---|
| trigger_name | character varying(64) | Trigger name |
| table_name | character varying(64) | Name of the table that defines the trigger |
| table_owner | character varying(64) | Owner of the table that defines the trigger |

18.3.293 USER_VIEWS

USER_VIEWS displays information about all views in the current schema.

| Column | Type | Description |
|-----------|-----------------------|-------------------|
| owner | character varying(64) | Owner of the view |
| view_name | character varying(64) | View name |

18.3.294 V\$SESSION

V\$SESSION displays all session information about the current session.

Table 18-348 V\$SESSION columns

| Name | Type | Description |
|------|--------|---|
| sid | Bigint | OID of the background process of the current activity |

| Name | Type | Description |
|----------|---------|---|
| serial# | Integer | Sequence number of the active background process, which is 0 in DWS. |
| user# | OID | OID of the user that has logged in to the background process |
| username | Name | Name of the user that has logged in to the background process |

18.3.295 V\$SESSION_LONGOPS

V\$SESSION_LONGOPS displays the progress of ongoing operations.

Table 18-349 V\$SESSION_LONGOPS columns

| Name | Type | Description |
|-----------|---------|--|
| sid | Bigint | OID of the running background process |
| serial# | Integer | Sequence number of the running background process, which is 0 in DWS. |
| sofar | Integer | Completed workload, which is empty in DWS. |
| totalwork | Integer | Total workload, which is empty in DWS. |

19 GUC Parameters of the DWS Database

19.1 Viewing GUC Parameters

DWS GUC parameters can control database system behaviors. You can check and adjust the GUC parameters based on your business scenario and data volume.

- Method 1: After a cluster is installed, you can check database parameters on the DWS management console.
- Method 2: You can also connect to a cluster and run SQL commands to check the GUC parameters.
 - Run the **SHOW** command.

NOTE

Method 2 can only be used to check the GUC parameter values of CNs, while the GUC parameter values of DN can be viewed through Method 1: by using the management console.

To view a certain parameter, run the following command:

```
SHOW server_version;
```

server_version indicates the database version.

Run the following command to view values of all parameters:

```
SHOW ALL;
```

- Use the **pg_settings** view.

To view a certain parameter, run the following command:

```
SELECT * FROM pg_settings WHERE NAME='server_version';
```

Run the following command to view values of all parameters:

```
SELECT * FROM pg_settings;
```

19.2 Configuring GUC Parameters

To ensure the optimal performance of DWS, you can adjust the GUC parameters in the database.

Parameter Types and Values

- The GUC parameters of DWS are classified into the following types:
 - **SUSET**: database administrator parameters. This type of parameters takes effect immediately after they are set. You do not need to restart the cluster. If a parameter of this type is set in the current session, the parameter takes effect only in the current session.
 - **USERSET**: common user parameters. This type of parameters takes effect immediately after they are set. You do not need to restart the cluster. If a parameter of this type is set in the current session, the parameter takes effect only in the current session.
 - **POSTMASTER**: database server parameters. This type of parameters takes effect only after the cluster is restarted. After you modify a parameter of this type, the system displays a message indicating that the cluster is to be restarted. You are advised to manually restart the cluster during off-peak hours for the setting to take effect.
 - **SIGHUP**: global database parameters. This type of parameters takes effect globally and cannot take effect for single sessions.
 - **BACKEND**: global database parameters. This type of parameters takes effect globally and cannot take effect for single sessions.
- All parameter names are case insensitive. A parameter value can be an integer, floating point number, string, Boolean value, or enumerated value.
 - The Boolean values can be **on/off**, **true/false**, **yes/no**, or **1/0**, and are case-insensitive.
 - The enumerated value range is specified in the **enumvals** column of the system catalog **pg_settings**.
- For parameters using units, specify their units during the setting, or default units are used.
 - The default units are specified in the **unit** column of **pg_settings**.
 - The unit of memory can be KB, MB, or GB.
 - The unit of time can be ms, s, min, h, or d.

Setting GUC Parameters

You can configure GUC parameters in the following ways:

- Method 1: After a cluster is created, log in to the DWS console and modify the database parameters of the cluster. For details, see "Modifying GUC Parameters of the DWS Cluster" in the *Data Warehouse Service (DWS) User Guide*.
- Method 2: Connect to a cluster and run SQL commands to configure the parameters of the SUSET or USERSET type.

Set parameters at database, user, or session levels.

- Set a database-level parameter.

```
ALTER DATABASE dbname SET paraname TO value;
```

The setting takes effect in the next session.
- Set a user-level parameter.

```
ALTER USER username SET paraname TO value;
```

The setting takes effect in the next session.

- Set a session-level parameter.

```
SET paraname TO value;
```

Parameter value in the current session is changed. After you exit the session, the setting becomes invalid.

Procedure

The following example shows how to set **explain_perf_mode**.

Step 1 View the value of **explain_perf_mode**.

```
SHOW explain_perf_mode;  
explain_perf_mode  
-----  
normal  
(1 row)
```

Step 2 Set **explain_perf_mode**.

Perform one of the following operations:

- Set a database-level parameter.

```
ALTER DATABASE gaussdb SET explain_perf_mode TO pretty;
```

If the following information is displayed, the setting has been modified.

```
ALTER DATABASE
```

The setting takes effect in the next session.

- Set a user-level parameter.

```
ALTER USER dbadmin SET explain_perf_mode TO pretty;
```

If the following information is displayed, the setting has been modified.

```
ALTER USER
```

The setting takes effect in the next session.

- Set a session-level parameter.

```
SET explain_perf_mode TO pretty;
```

If the following information is displayed, the setting has been modified.

```
SET
```

Step 3 Check whether the parameter is correctly set.

```
SHOW explain_perf_mode;  
explain_perf_mode  
-----  
pretty  
(1 row)
```

----End

19.3 GUC Parameter Usage

The database provides many operation parameters. Configuration of these parameters affects the behavior of the database system. Before modifying these parameters, learn the impact of these parameters on the database. Otherwise, unexpected results may occur.

Precautions

- If the value range of a parameter is a string, the string should comply with the naming conventions of the path and file name in the OS running the database.
- If the allowed maximum value of a parameter is **INT_MAX**, it indicates the maximum parameter value varies by OS.
- If the allowed maximum value of a parameter is **DBL_MAX**, it indicates the maximum parameter value varies by OS.

19.4 Connection and Authentication

19.4.1 Connection Settings

This section describes parameters related to the connection mode between the client and server.

max_connections

Parameter description: Specifies the maximum number of allowed parallel connections to the database. This parameter influences the concurrent processing capability of the cluster.

Type: POSTMASTER

Value range: an integer. For CNs, the value ranges from 100 to 16384. For DNs, the value ranges from 100 to 262143. Because there are internal connections in the cluster, the maximum value is rarely reached. If **invalid value for parameter "max_connections"** is displayed in the log, you need to decrease the **max_connections** value for DNs.

Default value: **800** for CNs and **5000** for DNs. If the default value is greater than the maximum value supported by kernel (determined when the **gs_initdb** command is executed), an error message will be displayed.

Setting suggestions:

Retain the default value of this parameter on CNs. On a DN, the value of this parameter is calculated as follows:

$\text{dop_limit} \times 20 \times 6 + 24$: **dop_limit** indicates the number of CPUs of each DN in the cluster. It is calculated as follows: **dop_limit** = Number of logical CPU cores of a single server/Number of DNs of a single server.

The minimum value is 5000.

If the parameter is set to a large value, DWS requires more SystemV shared memories or semaphores, which may exceed the maximum default configuration of the OS. In this case, modify the value as needed.

NOTICE

The value of **max_connections** is related to **max_prepared_transactions**. Before setting **max_connections**, ensure that the value of **max_prepared_transactions** is greater than or equal to that of **max_connections**. In this way, each session has a prepared transaction in the waiting state.

application_name

Parameter description: Specifies the name of the client program connecting to the database.

Type: USERSET

Value range: a string

Default value: `gsq`

connection_info

Parameter description: Specifies the database connection information, including the driver type, driver version, driver deployment path, and process owner. (This is an O&M parameter. Do not configure it by yourself.)

Type: USERSET

Value range: a string

Default value: an empty string

 NOTE

- An empty string indicates that the driver connected to the database does not support automatic setting of the **connection_info** parameter or the parameter is not set by users in applications.
- The following is an example of the concatenated value of **connection_info**:

```
{"driver_name":"ODBC","driver_version": "(GaussDB x.x.x build 39137c2d) compiled at 2022-09-23 15:43:11 commit 3629 last mr 5138 debug","driver_path":"/usr/local/lib/psqlodbcw.so","os_user":"omm"}
```

By default, **driver_name**, **driver_version**, **driver_path**, and **os_user** are displayed for ODBC, JDBC, and `gsq` connections. For other connections, **driver_name** and **driver_version** are displayed by default. The display of **driver_path** and **os_user** is controlled by users (see [Connecting to a Database](#) and [Configuring a Data Source in the Linux OS](#)).

19.4.2 Security and Authentication (postgresql.conf)

This section describes parameters about how to securely authenticate the client and server.

session_timeout

Parameter description: Specifies the maximum idle time without any operations after a connection to the server is established.

Type: USERSET

Value range: an integer ranging from 0 to 86400. The minimum unit is second (s). 0 means to disable the timeout.

Default value: 10 min

NOTICE

- The gsql client of DWS has an automatic reconnection mechanism. If the initialized local connection of a user to the server times out, gsql disconnects from and reconnects to the server.
 - Connections from the pooler connection pool to other CNs and DNPs are not controlled by the **session_timeout** parameter.
-

ssl_renegotiation_limit

Parameter description: Specifies the traffic volume over the SSL-encrypted channel before the session key is renegotiated. The renegotiation traffic limitation mechanism reduces the probability that attackers use the password analysis method to crack the key based on a huge amount of data but causes big performance losses. The traffic indicates the sum of sent and received traffic.

Type: USERSET

NOTE

You are advised to retain the default value, that is, disable the renegotiation mechanism. You are not advised to use the **gs_guc** tool or other methods to set the **ssl_renegotiation_limit** parameter in the **postgresql.conf** file. The setting does not take effect.

Value range: an integer ranging from 0 to **INT_MAX**. The unit is KB. 0 indicates that the renegotiation mechanism is disabled.

Default value: 0

failed_login_attempts

Parameter description: Specifies the maximum number of incorrect password attempts before an account is locked. The account will be automatically unlocked after the time specified in **password_lock_time**. For example, incorrect password attempts during login and password input failures when using the **ALTER USER** command

Type: SIGHUP

Value range: an integer ranging from 0 to 1000

- 0 indicates that the automatic locking function does not take effect.
- A positive number indicates that an account is locked when the number of incorrect password attempts reaches the value of **failed_login_attempts**.

Default value: 10

NOTICE

- The locking and unlocking functions take effect only when the values of **failed_login_attempts** and **password_lock_time** are positive numbers.
 - **failed_login_attempts** works with the SSL connection mode of the client to identify the number of incorrect password attempts. If PGSSLMODE is set to **allow** or **prefer**, two connection requests are generated for a password connection request. One request attempts an SSL connection, and the other request attempts a non-SSL connection. In this case, the number of incorrect password attempts perceived by the user is the value of **failed_login_attempts** divided by 2.
-

19.4.3 Communication Library Parameters

This section describes parameter settings and value ranges for communication libraries.

tcp_keepalives_idle

Parameter description: Specifies the interval between keepalive signal sending in an OS that supports the **TCP_KEEPIDLE** socket parameter. If no keepalive signal is transmitted, the connection is in idle state.

Type: USERSET

NOTICE

- If the OS does not support the **TCP_KEEPIDLE** parameter, set this parameter to 0.
 - The parameter is ignored on the OS where connections are established using the Unix domain socket.
-

Value range: an integer ranging from 0 to 3600. The unit is second (s).

Default value: 0

tcp_keepalives_interval

Parameter description: Specifies the response time before retransmission when the OS supports the **TCP_KEEPINTVL** socket parameter.

Type: USERSET

Value range: an integer ranging from 0 to 180. The unit is second (s).

Default value: 0

NOTICE

- If the OS does not support the **TCP_KEEPINTVL** parameter, set this parameter to **0**.
 - The parameter is ignored on the OS where connections are established using the Unix domain socket.
-

tcp_keepalives_count

Parameter description: Specifies the number of keepalived signals that can be waited before the DWS server is disconnected from the client if the OS supports the **TCP_KEEPCNT** socket parameter.

Type: USERSET

NOTICE

- If the OS does not support the **TCP_KEEPCNT** parameter, set this parameter to **0**.
 - The parameter is ignored on the OS where connections are established using the Unix domain socket.
-

Value range: an integer ranging from 0 to 100. **0** indicates that the connection is immediately broken if DWS does not receive a keepalived signal from the client.

Default value: 0

comm_max_datanode

Parameter description: Specifies the maximum number of DN's supported by the communication library.

Type: USERSET

Value range: an integer ranging from 1 to 8192

Default value: actual number of DN's

NOTICE

Increasing this parameter value takes effect immediately, while decreasing the value takes effect after the cluster is restarted.

comm_max_stream

Parameter description: maximum number of logical connection data structures cached in the communication library.

Type: SIGHUP

Value range: an integer ranging from 1 to 65535

Default value: 1024

 NOTE

If the value of `comm_max_datanode` is small, the process memory is sufficient. In this case, you can increase the value of `comm_max_stream`.

`max_stream_pool`

Parameter description: Specifies the maximum number of stream threads that can be contained in a stream thread pool. This feature is supported in 8.1.2 or later.

Type: SUSET

Value range: an integer ranging from -1 to INT_MAX. The values -1 and 0 indicate that the stream thread pool is disabled.

Default value:

- The formula for a new cluster is `max_stream_pool=MIN(max_connections, max_process_memory/16/5MB, 1024)`.
- The formula for a cluster upgraded from versions earlier than 8.3.0.100 is `max_stream_pool = MIN(max_connections, max_process_memory/16/5MB, 1024, value-of-the-old-cluster)`. During the upgrade, the settings for the new cluster are forcibly used. The old value is used if it is smaller.

 NOTE

- The number of stream threads in a thread pool can be reduced in real time. If the value of this parameter is increased, the number of stream threads is increased to meet the service requirements.
- Generally, you are advised not to change the value of this parameter because the stream thread pool supports the automatic cleanup function.
- If too many idle stream threads occupy the memory, you can decrease the value of this parameter to save the memory.

`enable_stream_sync_quit`

Parameter description: whether the stream threads exit synchronously when the stream plan ends. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that threads in the stream thread group exit after the stream plan ends.
- **off** indicates that stream threads exit directly after the stream plan ends without waiting for the threads in the stream thread group to exit.

Default value: off

enable_connect_standby

Parameter description: Sets the connection between a CN and a standby DN. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the CN connects to the standby server.
- **off** indicates that the CN connects to the primary DN.

Default value: off

CAUTION

- You are not advised to use this parameter in routine services. This parameter applies only to O&M operations. You are not advised to use the **gs_guc tool** for global settings. Otherwise, problems such as data inconsistency and result set errors may occur.
 - Enabling this parameter for a session with temporary tables will delete the temporary table data on DNs and prevent further actions on those tables.
-

comm_quota_size

Parameter description: Specifies the maximum size of packets that can be continuously sent by the communication library. When you use a 1GE NIC, a small value ranging from 20 KB to 40 KB is recommended.

Type: USERSET

Value range: an integer ranging from 0 to 102400. The default unit is KB. The value **0** indicates that the quota mechanism is not used.

Default value: 1 MB

comm_usable_memory

Parameter description: Specifies the maximum memory that can be used by the communication library cache on a single DN.

Type: SIGHUP

Value range: an integer ranging from 1 to 256. The default unit is KB. The minimum size cannot be less than 1 GB for installation.

Default value: max_process_memory/8

NOTICE

This parameter must be specifically set based on environment memory and the deployment method. If it is too large, out-of-memory (OOM) may occur. If it is too small, the performance of the communication library may deteriorate.

comm_client_bind

Parameter description: Specifies whether to bind the client of the communication library to a specified IP address when the client initiates a connection.

Type: USERSET

Value range: Boolean

- **on** indicates that the client is bound to a specified IP address.
- **off** indicates that the client is not bound to any IP addresses.

NOTICE

If multiple IP addresses of a node in a cluster are on the same communication network segment, set this parameter to **on**. In this case, the client is bound to the IP address specified by **listen_addresses**. The concurrency performance of a cluster depends on the number of random ports because a port can be used only by one client at a time.

Default value: off

comm_no_delay

Parameter description: Specifies whether to use the **NO_DELAY** attribute of the communication library connection. Restart the cluster for the setting to take effect.

Type: USERSET

Value range: Boolean

Default value: off

NOTICE

If packet loss occurs because a large number of packets are received per second, set this parameter to **off** to reduce the total number of packets.

comm_debug_mode

Parameter description: Specifies the debug mode of the communication library, that is, whether to print logs about the communication layer. The setting is effective at the session layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the detailed debug log of the communication library is printed.
- **off** indicates the detailed debug log of the communication library is not printed.

Default value: off

comm_ackchk_time

Parameter description: Specifies the duration after which the communication library server automatically triggers ACK when no data package is received.

Type: USERSET

Value range: an integer ranging from 0 to 20000. The unit is millisecond (ms). 0 indicates that automatic ACK triggering is disabled.

Default value: 2000

comm_timer_mode

Parameter description: Specifies the timer mode of the communication library, that is, whether to print timer logs in each phase of the communication layer. The setting is effective at the session layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the detailed timer log of the communication library is printed.
- **off** indicates the detailed timer log of the communication library is not printed.

Default value: off

comm_stat_mode

Parameter description: Specifies the stat mode of the communication library, that is, whether to print statistics about the communication layer. The setting is effective at the session layer.

NOTICE

When the switch is set to **on**, the number of printed logs is huge, adding extra overhead and reducing database performance. Therefore, set the switch to **on** only in the debug mode.

Type: USERSET

Value range: Boolean

- **on** indicates the statistics log of the communication library is printed.
- **off** indicates the statistics log of the communication library is not printed.

Default value: off

client_connection_check_interval

Parameter description: Specifies the interval for checking the client connection status. This parameter is supported by clusters of version 8.2.0 or later.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX. The unit is ms. The value 0 indicates that the client connection status is not checked.

Default value: 10000

NOTICE

During a long query executed in a session where a client (such as gsql, JDBC, or ODBC) directly connects to the CN,

- The CN checks the client connection status at the interval specified by **client_connection_check_interval**. If it detects that the client has been disconnected from the CN, the server terminates the long query and releases related resources to avoid waste of cluster resources.
- The DN checks its connection to the CN at the interval specified by **client_connection_check_interval**. If the DN detects that it has been disconnected from the CN, it terminates the long query and releases related resources to avoid waste of cluster resources.

conn_recycle_timeout

Parameter description: the interval for reclaiming idle connections between a CN and other nodes to the connection pool. This parameter is supported only by clusters of version 8.2.1 or later.

Type: USERSET

Value range: an integer ranging from 0 to 3600, in second (s). 0 indicates that the function of reclaiming idle connections is disabled.

Default value: 30

19.5 Resource Consumption

19.5.1 Memory

This section describes memory parameters.

NOTICE

Parameters described in this section take effect only after the database service restarts.

max_process_memory

Parameter description: Specifies the maximum physical memory of a database node.

Type: SIGHUP

Value range: an integer ranging from $2 \times 1024 \times 1024$ to $\text{INT_MAX}/2$. The unit is KB.

Default value: Determined based on non-secondary DNs. If multiple DNs are deployed on a server, the value is $(\text{Physical memory size}) \times 0.8 / (1 + \text{Number of primary DNs})$. If a single DN is deployed on a server, the value is $(\text{Physical memory size}) \times 0.6$. If the calculation result is less than 2 GB, the value is 2 GB by default. The default size of the secondary DN is 12 GB.

Setting suggestions:

 CAUTION

Single-node deployment applies only to existing users.

- On DNs, the value of this parameter is determined based on the server's physical memory and the number of DNs deployed on a single node.

If multiple DNs are deployed on a server, the calculation formula for the **max_process_memory** value is as follows: $(\text{Physical memory size} - \text{vm.min_free_kbytes}) \times 0.8 / (n + \text{Number of primary DNs})$. If only one DN is deployed on a server, the calculation formula for the **max_process_memory** value is $(\text{Physical memory size} - \text{vm.min_free_kbytes}) \times 0.6$.

This parameter aims to ensure system reliability, preventing node OOM caused by increasing memory usage. **vm.min_free_kbytes** indicates OS memory reserved for kernels to receive and send data. Its value is at least 5% of the total memory. That is, **max_process_memory** = $\text{Physical memory} \times 0.8 / (n + \text{Number of primary DNs})$. If the cluster scale (number of nodes in the cluster) is smaller than 256, $n=1$; if the cluster scale is larger than 256 and smaller than 512, $n=2$; if the cluster scale is larger than 512, $n=3$.

- You are not advised to set this parameter to the minimum threshold.
- Set this parameter on CNs to the same value as that on DN.
- RAM is the maximum memory allocated to the cluster.
- In DWS 8.2.0 and later versions, the initial value of **max_process_memory** is increased to improve memory resource utilization. However, in an unbalanced cluster where a server has two primary DNs running, using the initial value of **max_process_memory** may cause OOM. In DWS 8.2.0 and later versions, the **max_process_memory** parameter is changed to the SIGHUP type and can be manually adjusted. The **max_process_memory_auto_adjust** parameter is added. If a cluster is unbalanced, its CM will dynamically adjust **max_process_memory** based on the cluster status. The value of **max_process_memory** is $(\text{Physical memory} - \text{vm.min_free_kbytes}) \times 0.8 / \text{Number of primary DNs}$.
- In DWS 8.2.1 or later, the application scope of dynamically adjusting the value of **max_process_memory** is expanded from clusters where each server has only one DN to all cluster deployment modes.
 - If **max_process_memory_auto_adjust** is set to **on**, the value of **max_process_memory** is dynamically adjusted between the upper limit and the lower limit. The lower limit is calculated as follows: $(\text{Physical memory size}) \times 0.8 / (1 + \text{Number of primary DNs})$. The upper limit is specified by the GUC parameter **max_process_memory_balanced**. (For details about how to set **max_process_memory_balanced**, contact technical support.)
 - When the cluster works in load balancing mode, the upper limit of **max_process_memory** is used to improve the overall memory usage of the node. Compared with earlier versions, the memory usage is improved.
 - When the cluster is not in load balancing mode, the lower limit of **max_process_memory** is used. The overall memory usage of the node is the same as that in versions earlier than 8.2.1.
 - In upgrade scenarios, to ensure forward compatibility, the system does not set **max_process_memory_balanced**, and **max_process_memory** uses the value set before the upgrade by default.

max_process_memory_auto_adjust

Parameter description: Specifies whether to enable automatic adjustment for **max_process_memory** parameter. (This parameter is supported only by cluster versions 8.2.0 and later.)

Type: SIGHUP

Value range: Boolean

Default value: on

shared_buffers

Parameter description: Specifies the size of shared memory used by DWS. If this parameter is set to a large value, DWS may require more System V shared memory than the default setting.

Type: POSTMASTER

Value range: an integer ranging from 128 to INT_MAX. The unit is 8 KB.

Changing the value of **BLCKSZ** will result in a change in the minimum value of the **shared_buffers**.

Default value: The value of this parameter for CNs is half of that for DN, which is calculated using the formula **POWER(2,ROUND(LOG(2,max_process_memory/18),0))**. If the maximum value allowed by the OS is smaller than 32 MB, this parameter will be automatically changed to the maximum value allowed by the OS during database initialization.

Setting suggestions:

You are advised to set this parameter for DN to a value greater than that for CN, because DWS pushes its most queries down to DN.

It is recommended that **shared_buffers** be set to a value less than 40% of the memory. Set it to a large value for row-store tables and a small value for column-store tables. For column-store tables: **shared_buffers** = (Memory of a single server / Number of DN on the single server) x 0.4 x 0.25

If you want to increase the value of **shared_buffers**, you also need to increase the value of **checkpoint_segments**, because a longer period of time is required to write a large amount of new or changed data.

bulk_write_ring_size

Parameter description: Specifies the size of the ring buffer used for data parallel import.

Type: USERSET

Value range: an integer ranging from 16384 to INT_MAX. The unit is KB.

Default value: 2 GB

Setting suggestions: Increase the value of this parameter on DN if a huge amount of data is to be imported.

buffer_ring_ratio

Parameter description: ring buffer threshold for parallel data export

Type: USERSET

Value range: integer in the range 1–1000

Default value: 250

 NOTE

- The default value indicates that the threshold is 250/1000 (a quarter) of **shared_buffers**.
- The minimum value is 1/1000 of the value of **shared_buffers**.
- The maximum value is the value of **shared_buffers**.

Setting suggestions: If the cache hit ratio is not as expected during export, you are advised to configure this parameter on DN.

enable_cstore_ring_buffer

Parameter description: Specifies whether to enable column-store RingBuffer. This parameter is supported only by cluster versions 8.2.0 and later.

Type: USERSET

Value range: Boolean

Default value: off

Suggestion: If workloads have been running for a period of time, a large amount of frequently queried data has been stored in the CStoreBuffer, and you want to query large tables that are rarely accessed, you are advised to enable this function before the query and disable it after the query.

temp_buffers

Parameter description: Specifies the maximum size of local temporary buffers used by each database session.

Type: USERSET

Value range: an integer ranging from 800 to INT_MAX/2. The unit is 8 KB.

Default value: 8 MB

NOTE

- This parameter can be modified only before the first use of temporary tables within each session. Subsequent attempts to change the value of this parameter will not take effect on that session.
- Based on the value of **temp_buffers**, a session allocates temporary buffers as required. The cost of setting a large value in sessions that do not require many temporary buffers is only a buffer descriptor. If a buffer is used, 8192 bytes will be consumed for it.

max_prepared_transactions

Parameter description: Specifies the maximum number of transactions that can stay in the **prepared** state simultaneously. If this parameter is set to a large value, DWS may require more System V shared memory than the default setting.

When DWS is deployed as an HA system, set this parameter on the standby server to the same value or a value greater than that on the primary server. Otherwise, queries will fail on the standby server.

Type: POSTMASTER

Value range: an integer ranging from 0 to 536870911. The value of CN set to 0 indicates that the prepared transaction feature is disabled.

Default value: 800 for CNs and 5000 for DN

NOTE

Set this parameter to a value greater than or equal to that of **max_connections** to avoid failures in preparation.

work_mem

Parameter description: Specifies the memory capacity to be used by internal sort operations and Hash tables before writing to temporary disk files. Sort operations are used for **ORDER BY**, **DISTINCT**, and merge joins. Hash tables are required for Hash joins as well as Hash-based aggregations and **IN** subqueries.

For a complex query, several sort or Hash operations may be running in parallel; each operation will be allowed to use as much memory as this value specifies. If the memory is insufficient, data is written into temporary files. In addition, several running sessions could be performing such operations concurrently. Therefore, the total memory used may be many times the value of **work_mem**.

Type: USERSET

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: 512 MB for small-scale memory and 2 GB for large-scale memory (If **max_process_memory** is greater than or equal to 30 GB, it is large-scale memory. Otherwise, it is small-scale memory.)

Setting suggestions:

If the physical memory specified by **work_mem** is insufficient, additional operator calculation data will be written into temporary tables based on query characteristics and the degree of parallelism. This reduces performance by five to ten times, and prolongs the query response time from seconds to minutes.

- In complex serial query scenarios, each query requires five to ten associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/10.
- In simple serial query scenarios, each query requires two to five associated operations. Set **work_mem** using the following formula: **work_mem** = 50% of the memory/5.
- For concurrent queries, use the formula: **work_mem** = **work_mem** in serialized scenario/Number of concurrent SQL statements.

NOTICE

Once memory adaptation is enabled, there is no need to use **work_mem** to optimize operator memory usage after collecting statistics. The system generates a plan for each statement and estimates the memory usage of each operator and the entire statement based on the current workload. The system then schedules the queue based on the workload and the overall memory usage of the statement, which can result in statement queuing in high-concurrency scenarios.

query_mem

Parameter description: Specifies the memory used by query. If the value of **query_mem** is greater than 0, the optimizer adjusts the estimated query memory to this value when generating an execution plan.

Type: USERSET

Value range: 0 or an integer greater than 32 MB. The default unit is KB. If the value is set to a negative value or less than 32 MB, the default value 0 is used. In this case, the optimizer does not adjust the estimated query memory.

Default value: 0

query_max_mem

Parameter description: Specifies the maximum memory that can be used by query. If the value of **query_max_mem** is greater than 0, when generating an execution plan, the optimizer uses this value to set the available memory for operators. If job memory usage exceeds the value of this parameter, an error is reported and the job exits.

Type: USERSET

Value range: 0 or an integer greater than 32 MB. The default unit is KB. If the value is less than 32 MB, the system automatically sets this parameter to the default value 0. In this case, the optimizer does not limit the memory usage of jobs.

Default value: 0

agg_max_mem

Parameter description: Specifies the maximum memory that can be used by the Agg operator when the number of aggregation columns exceeds 5. This parameter takes effect only if the value of **agg_max_mem** is greater than 0. (This parameter is supported only in 8.1.3.200 and later cluster versions.)

Type: USERSET

Value range: 0 or an integer greater than 32 MB. The default unit is KB. If the value is less than 32 MB, the system automatically sets this parameter to the default value 0. In this case, the memory usage of the Agg operator is not limited based on the value.

Default value:

- If the current cluster is upgraded from an earlier version to 8.1.3 or later, the value in the earlier version is inherited. The default value is **INT_MAX**.
- If the current cluster version is 8.1.3 or later, the default value is **2GB**.

enable_rowagg_memory_control

Parameter description: Specifies the upper limit of the memory used by the row-store agg operator.

Type: USERSET

Value range: Boolean

- **on** indicates that the memory usage limit of the row-store agg operator is enabled. Setting this parameter to **on** can avoid OOM caused by the row-store agg operator, but may affect the agg performance.

- **off** indicates that the memory usage limit of the row-store agg operator is disabled. If this parameter is set to **off**, the system memory may be unavailable.

Default value: on

maintenance_work_mem

Parameter description: Specifies the maximum size of memory to be used for maintenance operations, such as **VACUUM**, **CREATE INDEX**, and **ALTER TABLE ADD FOREIGN KEY**. This parameter may affect the execution efficiency of VACUUM, VACUUM FULL, CLUSTER, and CREATE INDEX.

Type: USERSET

Value range: an integer ranging from 1024 to INT_MAX. The unit is KB.

Default value: 512 MB for small-scale memory and 2 GB for large-scale memory (If **max_process_memory** is greater than or equal to 30 GB, it is large-scale memory. Otherwise, it is small-scale memory.)

Setting suggestions:

- You are advised to set this parameter to the same value of **work_mem** so that database dump can be cleared or restored more quickly. In a database session, only one maintenance operation can be performed at a time. Maintenance is usually performed when there are not much sessions.
- When the **Automatic Cleanup** process is running, up to **autovacuum_max_workers** times of this memory may be allocated. Set **maintenance_work_mem** to a value equal to or larger than the value of **work_mem**.
- If a large amount of data needs to be processed in the cluster, increase the value of this parameter in sessions.

psort_work_mem

Parameter description: Specifies the memory used for internal sort operations on column-store tables before they are written into temporary disk files. This parameter can be used for inserting tables having a partial cluster key or index, creating a table index, and deleting or updating a table.

Type: USERSET

NOTICE

Multiple running sessions may perform partial sorting on a table at the same time. Therefore, the total memory usage may be several times of the **psort_work_mem** value.

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: 512 MB

max_loaded_cudesc

Parameter description: Specifies the number of loaded CuDescs per column when a column-store table is scanned. Increasing the value will improve the query performance and increase the memory usage, particularly when there are many columns in the column tables.

Type: USERSET

Value range: an integer ranging from 100 to INT_MAX/2

Default value: 1024

NOTICE

When the value of **max_loaded_cudesc** is set to a large value, the memory may be insufficient.

max_stack_depth

Parameter description: Specifies the maximum safe depth of DWS execution stack. The safety margin is required because the stack depth is not checked in every routine in the server, but only in key potentially-recursive routines, such as expression evaluation.

Type: SUSERSET

Take the following into consideration when setting this parameter:

- The ideal value of this parameter is the maximum stack size enforced by the kernel (value of **ulimit -s**).
- Setting this parameter to a value larger than the actual kernel limit means that a running recursive function may crash an individual backend process. In an OS where DWS can check the kernel limit, such as the SLES, DWS will prevent this parameter from being set to a value greater than the kernel limit.
- Since not all the OSs provide this function, you are advised to set a specific value for this parameter.

Value range: an integer ranging from 100 to INT_MAX. The unit is KB.

Default value: 2 MB

NOTE

2 MB is a small value and will not incur system breakdown in general, but may lead to execution failures of complex functions.

cstore_buffers

Parameter description: Specifies the size of the shared buffer used by ORC, Parquet, or CarbonData data of column-store tables and OBS or HDFS column-store foreign tables.

Type: POSTMASTER

Value range: an integer ranging from 16384 to INT_MAX. The unit is KB.

Default value: The value of this parameter for CNs is 32 MB, while that for DN is calculated using the formula **POWER(2,ROUND(LOG(2,max_process_memory/18),0))**.

Setting suggestions:

Column-store tables use the shared buffer specified by **cstore_buffers** instead of that specified by **shared_buffers**. When column-store tables are mainly used, reduce the value of **shared_buffers** and increase that of **cstore_buffers**.

Use **cstore_buffers** to specify the cache of ORC, Parquet, or CarbonData metadata and data for OBS or HDFS foreign tables. The metadata cache size should be 1/4 of **cstore_buffers** and not exceed 2 GB. The remaining cache is shared by column-store data and foreign table column-store data.

dfs_max_memory

Parameter description: Specifies the maximum memory that can be occupied during ORC export. If the memory is insufficient when a wide table is exported, increase the value of this parameter and try again. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: an integer ranging from 131072 to 10485760. The unit is KB.

Default value: 262144 KB (256 MB)

schedule_splits_threshold

Parameter description: Specifies the maximum number of files that can be stored in memory when you schedule an HDFS foreign table. If the number is exceeded, all files in the list will be spilled to disk for scheduling.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

Default value: 60000

bulk_read_ring_size

Parameter description: Specifies the ring buffer size used for data parallel export.

Type: USERSET

Value range: an integer ranging from 256 to INT_MAX. The unit is KB.

Default value: 16 MB

check_cu_size_threshold

Parameter description: When inserting data into a column-store table, if the amount of data already inserted in a CU exceeds the value of this parameter, row-level size verification will be performed to prevent the creation of uncompressed CUs larger than 1 GB.

Type: USERSET

Value range: an integer ranging from 0 to 1048576. The unit is KB.

Default value: 1 GB

NOTICE

If row-level size verification fails multiple times, you are advised to temporarily set the parameter to **0** at the session level.

memory_spread_strategy

Parameter description: Specifies the DN memory expansion policy of a customized resource pool. You are advised to set this parameter for services with sufficient memory. After this parameter is set, the query performance is improved. The maximum memory can be the same as that of the default resource pool. However, there may be errors caused by insufficient memory in some service scenarios if the memory is small. This parameter is supported only by clusters of version 8.1.3 or later.

Type: USERSET

Value range: enumerated values

- **none:** indicates that the memory is not expanded.
- **negative:** indicates that the memory and operators are expanded based on the estimated usage.
- **crazy:** indicates that the memory is directly expanded, which is equivalent to the memory expansion policy of the default resource pool. However, there may be errors caused by insufficient memory in some service scenarios if the memory is small.

Default value: none

19.5.2 Statement Disk Space Control

This section describes parameters related to statement disk space control, which are used to limit the disk space usage of statements.

sql_use_spacelimit

Parameter description: Specifies the allowed maximum space for files to be spilled to disks in a single SQL statement on a single DN. This parameter limits the space occupied by ordinary tables, temporary tables, and intermediate result sets spilled to disks. System administrators are also restricted by this parameter.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB. **-1** indicates no limit.

Default value: Set **sql_use_spacelimit** to 10% of the total disk space of the instance.

 NOTE

For example, if **sql_use_spacelimit** is set to **100** in the statement, and the data spilled to disks on a single DN exceeds 100 KB, DWS will stop the query and display a message indicating threshold exceeded.

```
insert into user1.t1 select * from user2.t1;  
ERROR: The space used on DN (104 kB) has exceeded the sql use space limit (100 kB).
```

Handling suggestion:

- Optimize the statement to reduce the data spilled to disks.
- If the disk space is sufficient, increase the value of this parameter.

temp_file_limit

Parameter description: Specifies the total space for files spilled to disks in a single thread. The temporary file can be the one used by sorting or hash tables, or cursors in a session.

This is a session-level setting.

Type: SUSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB. **-1** indicates no limit.

Default value: Set **temp_file_limit** to 10% of the total disk space of the instance.

NOTICE

This parameter does not apply to disk space occupied by temporary tablespaces used for executing SQL queries.

bi_page_reuse_factor

Parameter description: Specifies the percentage of idle space of old pages that can be reused when page replication is used for data synchronization between primary and standby DNs in the scenario where data is inserted into row-store tables in batches.

Type: USERSET

Value range: an integer ranging from 0 to 100. The value is a percentage. Value **0** indicates that the old pages are not reused and new pages are requested.

Default value: 70

NOTICE

- You are not advised to set this parameter to a value less than **50** (except **0**). If the idle space of the reused page is small, too much old page data will be transmitted between the primary and standby DNs. As a result, the batch insertion performance deteriorates.
- You are not advised to set this parameter to a value greater than **90**. If this parameter is set to a value greater than **90**, idle pages will be frequently queried, but old pages cannot be reused.

19.5.3 Kernel Resources

This section describes kernel resource parameters. Whether these parameters take effect depends on OS settings.

max_files_per_node

Parameter description: Specifies the maximum number of files that can be opened by a single SQL statement on a single node. Generally, you do not need to set this parameter.

Type: SUSET

Value range: an integer ranging from **-1** to **INT_MAX**. The value **-1** indicates that the maximum number is limited.

Default value: 50000

 NOTE

- For a newly installed cluster of 9.1.0 or later, the default value of this parameter is **50000**.
- In an upgrade scenario, if the original cluster supports the **max_files_per_node** parameter, the default value of this parameter remains the same for forward compatibility.
- In the upgrade scenario, if the original cluster does not support the **max_files_per_node** parameter, the default value of this parameter is **-1** after the upgrade.
- If error message "The last file name is [%s] and %d files have already been opened on data node [%s] with a maximum of %d files." is displayed during statement execution, increase the value of **max_files_per_node**.

19.5.4 Cost-based Vacuum Delay

The purpose of cost-based vacuum delay is to allow administrators to reduce the I/O impact of **VACUUM** and **ANALYZE** statements on concurrently active databases. For example, when maintenance statements such as **VACUUM** and **ANALYZE** do not need to be executed quickly and do not interfere with other database operations, administrators can use this function to achieve this purpose.

NOTICE

Certain operations hold critical locks and should be complete as quickly as possible. In DWS, cost-based vacuum delays do not take effect during such operations. To avoid uselessly long delays in such cases, the actual delay is calculated as follows and is the maximum value of the following calculation results:

- $\text{vacuum_cost_delay} * \text{accumulated_balance} / \text{vacuum_cost_limit}$
- $\text{vacuum_cost_delay} * 4$

During the execution of the ANALYZE | ANALYSE and VACUUM statements, the system maintains an internal counter that keeps track of the estimated cost of the various I/O operations that are performed. When the accumulated cost reaches a limit (specified by **vacuum_cost_limit**), the process performing the operation will sleep for a short period of time (specified by **vacuum_cost_delay**). Then, the counter resets and the operation continues.

By default, this feature is disabled. To enable this feature, set **vacuum_cost_delay** to a value other than 0.

vacuum_cost_delay

Parameter description: Specifies the length of time that the process will sleep when **vacuum_cost_limit** has been exceeded.

Type: USERSET

Value range: an integer ranging from 0 to 100. The unit is millisecond (ms). A positive number enables cost-based vacuum delay and **0** disables cost-based vacuum delay.

Default value: 0

NOTICE

- On many systems, the effective resolution of sleep length is 10 ms. Therefore, setting this parameter to a value that is not a multiple of 10 has the same effect as setting it to the next higher multiple of 10.
- This parameter is set to a small value, such as 10 or 20 milliseconds. Adjusting vacuum's resource consumption is best done by changing other parameters.

vacuum_cost_page_hit

Parameter description: Specifies the estimated cost for vacuuming a buffer found in the shared buffer. It represents the cost to lock the buffer pool, look up the shared Hash table, and scan the page.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

Default value: 1

vacuum_cost_page_miss

Parameter description: Specifies the estimated cost for vacuuming a buffer read from the disk. It represents the cost to lock the buffer pool, look up the shared Hash table, read the desired block from the disk, and scan the block.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

Default value: 2

vacuum_cost_page_dirty

Parameter description: Specifies the estimated cost charged when vacuum modifies a block that was previously clean. It represents the I/Os required to flush the dirty block out to disk again.

Type: USERSET

Value range: an integer ranging from 0 to 10000. The unit is millisecond (ms).

Default value: 20

vacuum_cost_limit

Parameter description: Specifies the cost limit. The cleanup process will sleep if this limit is exceeded.

Type: USERSET

Value range: an integer ranging from 1 to 10000. The unit is ms.

Default value: 200

19.5.5 Asynchronous I/O Operations

enable_adio_debug

Parameter description: Specifies whether to enable logging related to ADIO, which helps to locate ADIO-related issues. General users are not advised to set this O&M parameter.

Type: SUSET

Value range: Boolean

- **on** or **true** indicates the log switch is enabled.
- **off** or **false** indicates the log switch is disabled.

Default value: off

enable_fast_allocate

Parameter description: Specifies whether the quick allocation switch of the disk space is enabled. This switch can be enabled only in the XFS file system.

Type: SUSET

Value range: Boolean

- **on** or **true** indicates that this function is enabled.
- **off** or **false** indicates that the function is disabled.

Default value: off

prefetch_quantity

Parameter description: Specifies the number of row-store prefetches using the ADIO.

Type: USERSET

Value range: an integer ranging from 1024 to 1048576. The unit is 8 KB.

Default value: 32 MB

backwrite_quantity

Parameter description: Specifies the number of row-store writes using the ADIO.

Type: USERSET

Value range: an integer ranging from 1024 to 1048576. The unit is 8 KB.

Default value: 8MB

cstore_prefetch_quantity

Parameter description: Specifies the number of column-store prefetches using the ADIO.

Type: USERSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: 32 MB

cstore_backwrite_quantity

Parameter description: Specifies the number of column-store writes using the ADIO.

Type: USERSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: 8MB

cstore_backwrite_max_threshold

Parameter description: Specifies the maximum number of column-store writes buffered in the database using the ADIO.

Type: USERSET

Value range: an integer ranging from 4096 to INT_MAX/2, in KB

Default value: 2 GB

fast_extend_file_size

Parameter description: Specifies the disk size that the row-store pre-scales using the ADIO.

Type: SUSET

Value range: an integer. The value range is from 1024 to 1048576 and the unit is KB.

Default value: 8MB

effective_io_concurrency

Parameter description: Specifies the number of requests that can be simultaneously processed by the disk subsystem. For the RAID array, the parameter value must be the number of disk drive spindles in the array.

Type: USERSET

Value range: an integer ranging from 0 to 1000

Default value: 1

cu_preload_max_distance

Parameter description: Specifies the maximum number of CU groups that can be prefetched. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: an integer ranging from 0 to 1024. The value 0 indicates that prefetching is disabled.

Default value: 20

cu_preload_count

Parameter description: Specifies the maximum number of CUs to be prefetched. This parameter is supported only by clusters of version 9.1.0 or later.

Type: USERSET

Value range: an integer. The value ranges from 0 to 10000. The value 0 indicates that prefetching is disabled.

Default value: 600

19.5.6 Disk Caching

The following parameters are supported only by clusters of version 9.1.0 or later.

enable_disk_cache

Parameter description: Specifies whether to enable file caching. Setting this parameter to **on** only takes effect when **enable_aio_scheduler** is set to **on** and **obs_worker_pool_size** is greater than or equal to 4.

Type: USERSET

Value range: Boolean

Default value: off

enable_disk_cache_recovery

Parameter description: Specifies whether file caching can be restored when the cluster is restarted.

Type: USERSET

Value range: Boolean

Default value: off

19.6 Parallel Data Import

DWS provides a parallel data import function that enables a large amount of data to be imported in a fast and efficient manner. This section describes parameters for importing data in parallel in DWS.

raise_errors_if_no_files

Parameter description: Specifies whether distinguish between the problems "the number of imported file records is empty" and "the imported file does not exist". If set to **TRUE**, DWS reports the error "file does not exist" when the issue "the imported file does not exist" occurs.

Type: SUSET

Value range: Boolean

- **on** indicates the messages of "the number of imported file records is empty" and "the imported file does not exist" are distinguished when files are imported.
- **off** indicates the messages of "the number of imported file records is empty" and "the imported file does not exist" are not distinguished when files are imported.

Default value: off

partition_max_cache_size

Parameter description: To optimize the inserting of column-store partitioned tables in batches, data is cached during the inserting process and then written to the disk in batches. You can use **partition_max_cache_size** to specify the size of the data buffer. If the value is too large, much memory will be consumed. If it is

too small, the performance of inserting column-store partitioned tables in batches will deteriorate.

Type: USERSET

Value range: 4096 to INT_MAX/2. The minimum unit is KB.

Default value: 2 GB

partition_mem_batch

Parameter description: To optimize the performance of batch insert into column-store partitioned tables, data is cached during the inserting process and then written to the disk in batches. If **partition_max_cache_size** is configured, **partition_mem_batch** can be used to specify the number of caches. If this parameter is set to a large value, the available cache of each partition will be small, and the performance of batch insert into column-store partitioned tables will deteriorate. If this parameter is set to a small value, the available cache of each partition will be large, consuming much system memory.

Type: USERSET

Value range: 1 to 65535

Default value: 256

gds_debug_mod

Parameter description: Specifies whether to enable the debug function of General Data Service (GDS), which is a parallel data loading tool. This parameter is used to better locate and analyze GDS faults. After the debug function is enabled, types of packets received or sent by GDS, peer end of GDS during command interaction, and other interaction information about GDS are written into the logs of corresponding nodes. In this way, state switching on the GaussDB state machine and the current state are recorded. If this function is enabled, additional log I/O resources will be consumed, affecting log performance and validity. You are advised to enable this function only when locating GDS faults.

Type: USERSET

Value range: Boolean

- **on** indicates that the GDS debug function is enabled.
- **off** indicates that the GDS debug function is disabled.

Default value: off

max_copy_data_display

Parameter description: GUC control added for the length of the **rawrecord** field in the copy error table, in the text type. The maximum value is 1 GB minus 8203 bytes (that is, 1073733621 bytes). This parameter is supported only by clusters of version 8.2.1.100 or later.

When this parameter is set, it indicates the maximum number of characters that can be displayed. If the number of characters exceeds the maximum, an ellipsis (...) is displayed at the end.

Type: USERSET

Value range: 0 to 1073733616

Default value: 1024

enable_parallel_batch_insert

Parameter description: Specifies whether to enable the concurrent import function for row-store and column-store tables. This parameter is available only for clusters of version 8.2.0.102 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that concurrent import is enabled. Enabling the concurrent import function can effectively improve the execution speed of the INSERT, UPSERT, UPDATE, and DELETE statements for row-store and column-store tables.
- **off** indicates that concurrent import is **disabled**.

Default value: off

19.7 Write Ahead Logs

19.7.1 Settings

synchronous_commit

Parameter description: Specifies the synchronization mode of the current transaction.

Type: USERSET

Value range: enumerated values

- **on** indicates synchronization logs of a standby server are updated to disks.
- **off** indicates asynchronous commit.
- **local** indicates local commit.
- **remote_write** indicates synchronization logs of a standby server are written to disks.
- **remote_receive** indicates synchronization logs of a standby server are required to receive data.

Default value: on

wal_skip_fpw_level

Parameter description: Specifies whether to write the whole page to WALs when a page is modified for the first time after a checkpoint on a DN. This parameter controls if FPWs of different kinds are logged in WAL to avoid space shortage or slow synchronization between primary and standby nodes due to too many WAL

logs. This parameter is supported by cluster versions 8.3.0 and later. For versions earlier than 8.3.0, see **write_fpi_hint**.

Type: USERSET

Value range: an integer

- **0:** enables All FPWs.
- **1:** specifies that when setting hintbits for the heap tuples is the first change since checkpoint, FPW logs are not written. Whether this function takes effect is not affected by the **enable_crc_check** and **wal_log_hints** parameters. If this parameter is set to **1**, it is equivalent to **write_fpi_hint=off** in an earlier version. For details, see **write_fpi_hint**.
- **2:** FPW logs of indexes and FPW logs after the hintbits on the heap page is modified are not recorded.

Default value: 1

wal_decelerate_policy

Parameter description: Specifies the behavior policy after rate limiting is triggered. This parameter is supported only by clusters of 8.2.0 and later versions.

Type: USERSET

Value range: enumerated values

- **warning** indicates that an alarm is generated but the rate is not limited.
- **decelerate** indicates that the rate will be limited based on policy settings.

Default value: warning

NOTE

Setting the parameter to **warning** does not affect performance. Setting it to **decelerate** will limit the rate based on policy settings if the rate exceeds the threshold.

wal_write_speed

Parameter description: Specifies the maximum WAL write speed (byte/s) allowed by each query on a single DN. This parameter is supported only by clusters of 8.2.0 or later.

Type: USERSET

Value range: an integer ranging from 1024 to 10240000, in KB.

Default value: 30MB

NOTE

The rate of a large number of jobs with index copy and deletion operations will be limited.

wal_decelerate_trigger_threshold

Parameter description: Specifies the threshold of WAL write rate limiting for each query on a single DN. This parameter is supported only by cluster versions 8.2.0 and later.

Type: USERSET

Value range: an integer ranging from 1024 to 100000000000, in KB.

Default value: 128MB

 NOTE

This function is triggered only if the number of Xlogs generated by a single query is greater than the value of this parameter. DDL operations or a small number of DML operations are not affected.

commit_delay

Parameter description: Specifies the duration of committed data be stored in the WAL buffer.

Type: USERSET

Value range: an integer, ranging from 0 to 100000 (unit: μ s). **0** indicates no delay.

Default value: 0

NOTICE

- When this parameter is set to a value other than 0, the committed transaction is stored in the WAL buffer instead of being written to the WAL immediately. Then, the WalWriter process flushes the buffer out to disks periodically.
 - If system load is high, other transactions are probably ready to be committed within the delay. If no transactions are waiting to be submitted, the delay is a waste of time.
-

commit_siblings

Parameter description: Specifies a limit on the number of ongoing transactions. If the number of ongoing transactions is greater than the limit, a new transaction will wait for the period of time specified by [commit_delay](#) before it is submitted. If the number of ongoing transactions is less than the limit, the new transaction is immediately written into a WAL.

Type: USERSET

Value range: an integer ranging from 0 to 1000

Default value: 5

wal_compression

Parameter description: Specifies whether to compress FPI pages.

Type: USERSET

Value range: Boolean

- **on:** enable the compression

- **off**: disable the compression

Default value: on

NOTICE

- Only zlib compression algorithm is supported.
 - For clusters that are upgraded to the current version from an earlier version, this parameter is set to **off** by default. You can run the **gs_guc** command to enable the FPI compression function if needed.
 - If the current version is a newly installed version, this parameter is set to **on** by default.
 - If this parameter is manually enabled for a cluster upgraded from an earlier version, the cluster cannot be rolled back.
-

wal_compression_level

Parameter description: Specifies the compression level of zlib compression algorithm when the **wal_compression** parameter is enabled.

Type: USERSET

Value range: an integer ranging from 0 to 9.

- **0** indicates no compression.
- **1** indicates the lowest compression ratio.
- **9** indicates the highest compression ratio.

Default value: 9

19.7.2 Checkpoints

checkpoint_segments

Parameter description: minimum number of WAL segment files in the period specified by **checkpoint_timeout**. The size of each log file is 16 MB.

Type: SIGHUP

Value range: an integer. The minimum value is 1.

Default value: 64

NOTICE

Increasing the value of this parameter speeds up the import of big data. Set this parameter based on **checkpoint_timeout** and **shared_buffers**. This parameter affects the number of WAL log segment files that can be reused. Generally, the maximum number of reused files in the **pg_xlog** folder is twice the number of checkpoint segments. The reused files are not deleted and are renamed to the WAL log segment files which will be later used.

19.8 HA Replication

19.8.1 Primary Server

`enable_data_replicate`

Parameter description: Specifies the data synchronization mode between the primary and standby servers when data is imported to row-store tables in a database.

Type: USERSET

Value range: Boolean

- **on** indicates that data pages are used for the data synchronization between the primary and standby servers when data is imported to row-store tables in a database. This parameter cannot be set to **on** if **replication_type** is set to **1**.
- **off** indicates that the primary and standby servers synchronize data using Xlogs while the data is imported to a row-store table.

Default value: on

`ha_module_debug`

Parameter description: This command is used to view the replication status log of a specific data block during data replication.

Type: USERSET

Value range: Boolean

- **on** indicates that the log records the status of each data block during data replication.
- **off** indicates that the status of each data block is not recorded in logs during data replication.

Default value: off

19.9 Query Planning

19.9.1 Optimizer Method Configuration

These configuration parameters provide a crude method of influencing the query plans chosen by the query optimizer. If the default plan chosen by the optimizer for a particular query is not optimal, a temporary solution is to use one of these configuration parameters to force the optimizer to choose a different plan. Better ways include adjusting the optimizer cost constants, manually running **ANALYZE**, increasing the value of the [default_statistics_target](#) configuration parameter, and adding the statistics collected in a specific column using **ALTER TABLE SET STATISTICS**.

enable_bitmapscan

Parameter description: Controls whether the query optimizer uses the bitmap-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_hashagg

Parameter description: Controls whether the query optimizer uses the Hash aggregation plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_mixedagg

Parameter description: Controls whether the query optimizer uses the Mixed Agg plan type.

Type: USERSET

Value range: Boolean

- **on** indicates that a Mixed Agg query plan is generated for the Grouping Sets statement (including Rollup or Cube) that meets certain conditions.
- **off** indicates it is disabled.

Default value: on

NOTICE

- The default value of this parameter is **on** in a newly installed cluster of 9.1.0.200 or later. In an upgrade scenario, the default value of this parameter is retained for forward compatibility.
- The Mixed Agg query plan can be used to improve the performance of statements dealing with a large amount of data (the data volume of a single DN table is greater than 100 GB).

Mixed Agg is not supported in the following scenarios:

- The data type of the columns in the **GROUP BY** clause do not support hashing.
- The aggregate function uses **DISTINCT** for deduplication or **ORDER BY** for sorting.
- The **GROUPING SETS** clause does not contain empty groups.

enable_hashjoin

Parameter description: Controls whether the query optimizer uses the Hash-join plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_indexscan

Parameter description: Controls whether the query optimizer uses the index-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_indexonlyscan

Parameter description: Controls whether the query optimizer uses the index-only-scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.

- **off** indicates it is disabled.

Default value: on

enable_material

Parameter description: Controls whether the query optimizer uses materialization. It is impossible to suppress materialization entirely, but setting this parameter to **off** prevents the optimizer from inserting materialized nodes.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_mergejoin

Parameter description: Controls whether the query optimizer uses the merge-join plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: off

enable_nestloop

Parameter description: Controls whether the query optimizer uses the nested-loop join plan type to fully scan internal tables. It is impossible to suppress nested-loop joins entirely, but setting this parameter to **off** allows the optimizer to choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: off

enable_index_nestloop

Parameter description: Controls whether the query optimizer uses the nested-loop join plan type to scan the parameterized indexes of internal tables.

Type: USERSET

Value range: Boolean

- **on** indicates the query optimizer uses the nested-loop join plan type.
- **off** indicates the query optimizer does not use the nested-loop join plan type.

Default value: The default value for a newly installed cluster is **on**. If the cluster is upgraded from R8C10, the forward compatibility is retained. If the version is upgraded from R7C10 or an earlier version, the default value is **off**.

left_join_estimation_enhancement

Parameter description: Specifies whether to use the optimized estimated number of rows for left join. This parameter is supported only by clusters of version 8.3.0.100 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimized value is used.
- **off** indicates it is disabled.

Default value: off

enable_seqscan

Parameter description: Controls whether the query optimizer uses the sequential scan plan type. It is impossible to suppress sequential scans entirely, but setting this variable to **off** allows the optimizer to preferentially choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_sort

Parameter description: Controls whether the query optimizer uses the sort method. It is impossible to suppress explicit sorts entirely, but setting this variable to **off** allows the optimizer to preferentially choose other methods if available.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

max_opt_sort_rows

Parameter description: Specifies the maximum number of optimized limit+offset rows in an ORDER BY clause. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

- If the value is 0, the parameter does not take effect.
- If this parameter is set to any other value, the optimization takes effect when the number of limit+offset rows in the ORDER BY clause is less than the value of this parameter. If the number of limit+offset rows in the order by clause is greater than the value of this parameter, the optimization does not take effect. After the optimization, the time required is reduced, but the memory usage may increase.

Default value: 0

enable_tidscan

Parameter description: Controls whether the query optimizer uses the Tuple ID (TID) scan plan type.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_kill_query

Parameter description: In CASCADE mode, when a user is deleted, all the objects belonging to the user are deleted. This parameter specifies whether the queries of the objects belonging to the user can be unlocked when the user is deleted.

Type: SUSERSET

Value range: Boolean

- **on** indicates the unlocking is allowed.
- **off** indicates the unlocking is not allowed.

Default value: off

enforce_oracle_behavior

Parameter description: Controls the rule matching modes of regular expressions.

Type: USERSET

Value range: Boolean

- **on** indicates that the ORACLE matching rule is used.
- **off** indicates that the POSIX matching rule is used.

Default value: on

enable_stream_concurrent_update

Parameter description: Controls the use of **stream** in concurrent updates. This parameter is restricted by the [enable_stream_operator](#) parameter.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimizer can generate stream plans for the **UPDATE** statement.
- **off** indicates that the optimizer can generate only non-stream plans for the **UPDATE** statement.

Default value: on

enable_stream_ctescan

Parameter description: Specifies whether a stream plan supports **ctescan**.

Type: USERSET

Value range: Boolean

- **on** indicates that **ctescan** is supported for the stream plan.
- **off** indicates that **ctescan** is not supported for the stream plan.

Default value: on

NOTE

In upgrade scenarios, the default value of this parameter is forward compatible, and the original value is retained.

enable_stream_operator

Parameter description: Controls whether the query optimizer uses streams.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_stream_recursive

Parameter description: Specifies whether to push **WITH RECURSIVE** join queries to DNs for processing.

This parameter is discarded in 8.2.1.210 and later cluster versions. To ensure forward compatibility, the setting can be successful but does not take effect. You are advised to use the [stream_recursive_option](#) parameter.

Type: USERSET

Value range: Boolean

- **on:** **WITH RECURSIVE** join queries will be pushed down to DNs.
- **off:** **WITH RECURSIVE** join queries will not be pushed down to DNs.

Default value: on

stream_recursive_option

Parameter description: Specifies whether to push **WITH RECURSIVE** join queries to DNs for processing. This parameter is supported only by 8.2.1.210 and later cluster versions.

Type: USERSET

Value range: enumerated values

- **none:** **WITH RECURSIVE** queries will not be pushed down to DNs.
- **normal:** **WITH RECURSIVE** join queries can be pushed down to DNs for distributed execution in single-logical cluster scenarios.
- **nodegroup:** **WITH RECURSIVE** join queries can be pushed down to DNs for distributed execution in multi-logical cluster scenarios.

Default value: normal

enable_value_redistribute

Parameter description: Specifies whether to generate value redistribute plans. In 8.2.0 and later cluster versions, this parameter takes effect for **rank**, **dense_rank**, and **row_number** without the **PARTITION BY** clause.

Type: USERSET

Value range: Boolean

- **on** indicates that value redistribute plans are generated.
- **off** indicates that no value redistribute plans are generated.

Default value: on

max_recursive_times

Parameter description: Specifies the maximum number of **WITH RECURSIVE** iterations.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 200

enable_vector_engine

Parameter description: Controls whether the query optimizer uses the vectorized executor.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_broadcast

Parameter description: Controls whether the query optimizer uses the broadcast distribution method when it evaluates the cost of stream.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

enable_redistribute

Parameter description: Controls whether the query optimizer uses the local redistribute or split redistribute distribution method when estimating the cost of streams. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that either of the distribution methods is used.
- **off** indicates that none of the distribution methods is used.

Default value: on

enable_change_hjcost

Parameter description: Specifies whether the optimizer excludes internal table running costs when selecting the Hash Join cost path. If it is set to **on**, tables with a few records and high running costs are more possible to be selected.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: off

enable_fstream

Parameter description: Controls whether the query optimizer uses streams when it delivers statements. This parameter is only used for external HDFS tables.

This parameter has been discarded. To reserve forward compatibility, set this parameter to **on**, but the setting does not make a difference.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: off

enable_hashfilter

Parameter description: Controls whether hashfilters can be generated for plans that contain replication tables (including dual and constant tables). This parameter is supported by clusters of version 8.2.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that hashfilters can be generated.
- **off** indicates that no hashfilters can be generated.

Default value: on

best_agg_plan

Parameter description: The query optimizer generates three plans for the aggregate operation under the stream:

1. hashagg+gather(redistribute)+hashagg
2. redistribute+hashagg(+gather)
3. hashagg+redistribute+hashagg(+gather).

This parameter is used to control the query optimizer to generate which type of hashagg plans.

Type: USERSET

Value range: an integer ranging from 0 to 3.

- When the value is set to **1**, the first plan is forcibly generated.
- When the value is set to **2** and if the **group by** column can be redistributed, the second plan is forcibly generated. Otherwise, the first plan is generated.
- When the value is set to **3** and if the **group by** column can be redistributed, the third plan is generated. Otherwise, the first plan is generated.
- When the value is set to **0**, the query optimizer chooses the most optimal plan based on the estimated costs of the three plans above.

Default value: 0

turbo_engine_version

Parameter description: For tables with the turbo storage format specified during table creation (by setting the **enable_turbo_store** parameter to **on** in the table properties), and when the query does not involve merge join or sort agg operators, the executor can use the turbo execution engine, which can significantly improve performance.

Type: USERSET

Value range: an integer ranging from 0 to 3.

- The value **0** indicates that the turbo execution engine is disabled.
- The value **1** indicates that the turbo execution engine is only used for single-table aggregate queries.
- The value **2** indicates that the turbo execution engine is only used for single-table aggregate or multi-table join queries.
- The value **3** indicates that the turbo execution engine can be used to accelerate most commonly used operators, except for operators such as merge join and sort agg. When the data volume is large and **turbo_engine_version** is set to **3**, the occurrence of merge join and sort agg operators is relatively rare, so turbo execution engine acceleration can be achieved for almost SQL statements.

Default value: 3

enable_bucket_stream_opt

Parameter description: Specifies whether to use the **bucket agg** and **bucket join** policies for level-2 partitioned tables or 3.0 hash distributed tables. It speeds up SQL statement execution by avoiding local data redistribution or broadcast. This is supported only by clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: Boolean

- **true:** The optimizer uses the **bucket agg** and **bucket join** execution policies to generate plans when the conditions for the policy to be applied are met. If this optimization policy is used, "Bucket Stream: true" is displayed at the end of the **EXPLAIN** statement.
- **false:** The optimizer does not use the **bucket agg** and **bucket join** execution policies to generate plans.

Default value: true

NOTICE

- The default value of this parameter is **true** in a newly installed cluster of 9.1.0.200 or later. In an upgrade scenario, the default value of this parameter is retained for forward compatibility.
- The **bucket agg** and **bucket join** execution policies take effect only when the current query has 16 or fewer available CPUs and meets one of the following conditions:
 1. The distribution column for level-2 partitions must match the **secondary_part_column** of these partitions. It is recommended that the number of level-2 partitions be the number of DN's multiplied by 12. Supported multiples include 4, 6, 8, 12, and 16.
 2. Tables in version 3.0 must use hash distribution, with the number of buckets or DN's exceeding 10.
- If the local stream cost in the plan is low, the query may not select the **bucket agg** and **bucket join** policies.

spill_compression

Parameter description: Specifies the compression algorithm used when the executor operator runs out of memory and needs to spill data to disk. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: enumerated values

- 'lz4' indicates that the lz4 compression algorithm is used, which provides better performance for scenarios with smaller spill volumes, but requires more storage space.
- 'zstd' indicates that the zstd compression algorithm is used, which provides better performance for scenarios with larger spill volumes where I/O is the main bottleneck, and requires approximately 2/3 of the storage space used by lz4.

Default value: 'lz4'

index_selectivity_cost

Parameter description: Controls the cost calculation of cbtree when scanning column-store table indexes (for selectivity > 0.001). This parameter is only supported by clusters of version 8.2.1.100 or later.

Type: USERSET

Value range: a floating point number, which can be -1 or ranges from 0 to 1000.

- If this parameter is set to **0**, the index selection rate is not affected by the threshold 0.001.
- If the value is **-1**, the value is impacted by **disable_cost**.
- When it is set to other values, the value is the coefficient for cbtree cost calculation.

Default value: -1

index_cost_limit

Parameter description: threshold for disabling the cost calculation of cbtree during column-store table index scanning. This parameter is supported only by clusters of version 8.2.1.100 or later.

Type: USERSET

Value range: an integer ranging from 0 to 2147483647

- If the value is **0**, the parameter does not take effect.
- If this parameter is set to other values and the number of rows in a table is less than the value of this parameter, the table is not affected by the index selection rate threshold 0.001.

Default value: 0

volatile_shipping_version

Parameter description: Controls the execution scope of volatile functions to be pushed down.

Type: USERSET

Value range: 0, 1, 2, 3

- When set to **3**, it extends the support for pushing down InlineCTE when it is only referenced once, on top of the support provided by a value **2**. It also extends the support for pushing down the use of volatile functions in UPSERT operations involving replicated tables.
- When the value is **2**, pushdown can be performed when VOLATILE functions are contained in the target column of the copied CTE result.
- If this parameter is set to **1**, the **nextval**, **uuid_generate_v1**, **sys_guid**, and **uuid** functions can be completely pushed down if they are in the target column of a statement.
- If this parameter is set to **0**, random functions can be completely pushed down. The **nextval** and **uuid_generate_v1** functions can be pushed down only if **INSERT** contains simple query statements.

Default value: 3

agg_redistribute_enhancement

Parameter description: When the aggregate operation is performed, which contains multiple **group by** columns and all of the columns are not in the distribution column, you need to select one **group by** column for redistribution. This parameter controls the policy of selecting a redistribution column.

Type: USERSET

Value range: Boolean

- **on** indicates the column that can be redistributed and evaluates the most distinct value for redistribution.

- **off** indicates the first column that can be redistributed for redistribution.

Default value: off

enable_valuepartition_pruning

Parameter description: Specifies whether to perform static or dynamic optimization on the partitioned tables in a distributed file system (DFS).

Type: USERSET

Value range: Boolean

- **on** indicates that the DFS partitioned table is dynamically or statically optimized.
- **off** indicates that the DFS partitioned table is not dynamically or statically optimized.

Default value: on

expected_computing_nodegroup

Parameter description: Specifies a computing Node Group or the way to choose such a group. The Node Group mechanism is now for internal use only. You do not need to set it.

During join or aggregation operations, a Node Group can be selected in four modes. In each mode, the specified candidate computing Node Groups are listed for the optimizer to select an appropriate one for the current operator.

Type: USERSET

Value range: a string

- **optimal:** The list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the DNs in the Node Groups on which the current user has the COMPUTE permission.
- **query:** The list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the DNs in the Node Groups where base tables involved in the query are located.
- **bind:** If the current session user is a logical cluster user, the candidate computing Node Group is the Node Group of the logical cluster associated with the current user. If the session user is not a logical cluster user, the candidate computing Node Group selection rule is the same as that when this parameter is set to **query**.
- Node Group name:
 - If [enable_nodegroup_debug](#) is set to **off**, the list of candidate computing Node Groups consists of the Node Group where the operator's operation objects are located and the specified Node Group.
 - If [enable_nodegroup_debug](#) is set to **on**, the specified Node Group is used as the candidate Node Group.

Default value: bind

enable_nodegroup_debug

Parameter description: Specifies whether the optimizer assigns computing workloads to a specific Node Group when multiple Node Groups exist in an environment. The Node Group mechanism is now for internal use only. You do not need to set it.

This parameter takes effect only when **expected_computing_nodegroup** is set to a specific Node Group.

Type: USERSET

Value range: Boolean

- **on** indicates that computing workloads are assigned to the Node Group specified by **expected_computing_nodegroup**.
- **off** indicates no Node Group is specified to compute.

Default value: off

stream_multiple

Parameter description: Specifies the weight used for optimizer to calculate the final cost of stream operators.

The base stream cost is multiplied by this weight to make the final cost.

Type: USERSET

Value range: a floating point number ranging from 0 to 10000

Default value: 1

NOTICE

This parameter is applicable only to Redistribute and Broadcast streams.

qrw_inlist2join_optmode

Parameter description: Specifies whether enable inlist-to-join (inlist2join) query rewriting.

Type: USERSET

Value range: a string

- **disable:** inlist2join disabled
- **cost_base:** cost-based inlist2join query rewriting
- **rule_base:** forcible rule-based inlist2join query rewriting
- A positive integer: threshold of Inlist2join query rewriting. If the number of elements in the list is greater than the threshold, the rewriting is performed.

Default value: disable

enable_inlist_hashing

Parameter description: Specifies whether to use inlist hash optimization. This parameter is supported only by clusters of version 9.1.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that inlist hash optimization is enabled.
- **off** indicates that inlist hash optimization is disabled.

Default value: on

setop_optmode

Parameter description: Specifies whether to perform deduplication on the query branch statements of a set operation (**UNION/EXCEPT/INTERSECT**) without the **ALL** option.

Type: USERSET

Value range: enumerated values

- **disable:** The query branch does not perform deduplication.
- **force:** The query branch forcibly performs deduplication.
- **cost:** The optimizer evaluates the costs of query branches with and without deduplication and selects the execution mode with the lower cost.

Default value: cost

NOTICE

- The default value of this parameter is **cost** in a newly installed cluster of 9.1.0.200 or later. In an upgrade scenario, the default value of this parameter is retained for forward compatibility.
 - This parameter takes effect only if the execution plan of a SQL statement meets the following conditions:
 - The **UNION**, **EXCEPT**, and **INTERSECT** operations in the SQL statement do not contain the **ALL** option.
 - Data redistribution has been performed on the query branches where the set operation is to be performed.
-

skew_option

Parameter description: Specifies whether an optimization policy is used

Type: USERSET

Value range: a string

- **off:** policy disabled
- **normal:** radical policy. All possible skews are optimized.

- **lazy**: conservative policy. Uncertain skews are ignored.

Default value: normal

enable_expr_skew_optimization

Parameter description: Specifies whether to use expression statistics in the skew optimization policy. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that expression statistics are used to determine whether data skew occurs in the skew optimization policy.
- **off** indicates that expression statistics are not used to determine whether data skew occurs in the skew optimization policy.

Default value: on

prefer_hashjoin_path

Parameter description: whether to preferentially generate hashjoin paths so that other paths with high costs can be pre-pruned to shorten the overall plan generation time. This parameter is supported only by clusters of version 8.2.1 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimization of generating hash join paths in advance is enabled.
- **off** indicates that the optimization of generating hash join paths in advance is disabled.

Default value: on

enable_hashfilter_test

Parameter description: whether to add hash filters to columns for base table scan to check whether the results meet expectations. In addition, this parameter determines whether to check the DN accuracy when data is inserted (that is, whether the current data should be inserted into the current DN).

Type: USERSET

Value range: Boolean

- **on** adds a hash filter for the distribution column to the base table scan and performs accurate DN verification during data insertion.
- **off** does not add a hash filter for the distribution column to the base table scan and does not perform DN verification during data insertion.

Default value: on

NOTICE

- This parameter is valid only for tables distributed in hash mode.
- If this parameter is set to **on**, DN accuracy is verified during data insertion, affecting data insertion performance.

enable_cu_align_8k

Parameter description: Specifies whether to set the CUs in V3 tables to 8 KB. This parameter is supported only by clusters of version 9.1.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the CUs in V3 tables are set to 8,192 bytes.
- **off** indicates that the CUs in V3 tables are set to 512 bytes.

Default value: off

enable_cu_batch_insert

Parameter description: Specifies whether to enable the multi-column CU batch write feature for V2 tables. This parameter is supported only by clusters of version 9.1.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the multi-column CU batch write feature is enabled for V2 tables.
- **off** indicates that the multi-column CU batch write feature is disabled for V2 tables.

Default value: off

enable_topk_optimization

Parameter description: Specifies whether to enable Top K sorting optimization. This is supported only by clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that Top K sorting optimization is enabled.
- **off** indicates that Top K sorting optimization is disabled.

Default value: on

19.9.2 Optimizer Cost Constants

This section describes the optimizer cost constants. The cost variables described in this section are measured on an arbitrary scale. Only their relative values matter,

therefore scaling them all in or out by the same factor will result in no differences in the optimizer's choices. By default, these cost variables are based on the cost of sequential page fetches, that is, **seq_page_cost** is conventionally set to **1.0** and the other cost variables are set with reference to the parameter. However, you can use a different scale, such as actual execution time in milliseconds.

seq_page_cost

Parameter description: Specifies the optimizer's estimated cost of a disk page fetch that is part of a series of sequential fetches.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 1

random_page_cost

Parameter description: Specifies the optimizer's estimated cost of an out-of-sequence disk page fetch.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 4

NOTE

- Although the server allows you to set the value of **random_page_cost** to less than that of **seq_page_cost**, it is not physically sensitive to do so. However, setting them equal makes sense if the database is entirely cached in RAM, because in that case there is no penalty for fetching pages out of sequence. Also, in a heavily-cached database you should lower both values relative to the CPU parameters, since the cost of fetching a page already in RAM is much smaller than it would normally be.
- This value can be overwritten for tables and indexes in a particular tablespace by setting the tablespace parameter of the same name.
- Comparing to **seq_page_cost**, reducing this value will cause the system to prefer index scans and raising it makes index scans relatively more expensive. You can increase or decrease both values at the same time to change the disk I/O cost relative to CPU cost.

cpu_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each row during a query.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 0.01

cpu_index_tuple_cost

Parameter description: Specifies the optimizer's estimated cost of processing each index entry during an index scan.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 0.005

cpu_operator_cost

Parameter description: Specifies the optimizer's estimated cost of processing each operator or function during a query.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 0.0025

effective_cache_size

Parameter description: Specifies the optimizer's assumption about the effective size of the disk cache that is available to a single query.

When setting this parameter you should consider both DWS's shared buffer and the kernel's disk cache. Also, take into account the expected number of concurrent queries on different tables, since they will have to share the available space.

This parameter has no effect on the size of shared memory allocated by DWS. It is used only for estimation purposes and does not reserve kernel disk cache. The value is in the unit of disk page. Usually the size of each page is 8192 bytes.

Type: USERSET

Value range: an integer ranging is from 1 to INT_MAX. The unit is 8 KB.

A value greater than the default one may enable index scanning, and a value less than the default one may enable sequence scanning.

Default value: 128MB

allocate_mem_cost

Parameter description: Specifies the query optimizer's estimated cost of creating a Hash table for memory space using Hash join. This parameter is used for optimization when the Hash join estimation is inaccurate.

Type: USERSET

Value range: a floating point number ranging from 0 to DBL_MAX

Default value: 0

smp_thread_cost

Parameter description: Specifies the optimizer's cost for calculating parallel threads of an operator. This parameter is used for tuning if **query_dop** is not suitable for system load management. (This parameter is supported only by clusters of version 8.2.0 or later.)

Type: USERSET

Value range: a floating point number ranging from 1 to 10000

Default value: 1000

19.9.3 Genetic Query Optimizer

This section describes parameters related to genetic query optimizer. The genetic query optimizer (GEQO) is an algorithm that plans queries by using heuristic searching. This algorithm reduces planning time for complex queries and the cost of producing plans are sometimes inferior to those found by the normal exhaustive-search algorithm.

geqo

Parameter description: Controls the use of genetic query optimization.

Type: USERSET

Value range: Boolean

- **on** indicates GEQO is enabled.
- **off** indicates GEQO is disabled.

Default value: on

NOTICE

Generally, do not set this parameter to **off**. **geqo_threshold** provides more subtle control of GEQO.

geqo_threshold

Parameter description: Specifies the number of **FROM** items. Genetic query optimization is used to plan queries when the number of statements executed is greater than this value.

Type: USERSET

Value range: an integer ranging from 2 to INT_MAX

Default value: 12

NOTICE

- For simpler queries it is best to use the regular, exhaustive-search planner, but for queries with many tables it is better to use GEQO to manage the queries.
 - A **FULL OUTER JOIN** construct counts as only one **FROM** item.
-

geqo_effort

Parameter description: Controls the trade-off between planning time and query plan quality in GEQO.

Type: USERSET

Value range: an integer ranging from 1 to 10

Default value: 5

NOTICE

- Larger values increase the time spent in query planning, but also increase the probability that an efficient query plan is chosen.
 - **geqo_effort** does not have direct effect. This parameter is only used to compute the default values for the other variables that influence GEQO behavior. You can manually set other parameters as required.
-

geqo_pool_size

Parameter description: Specifies the pool size used by GEQO, that is, the number of individuals in the genetic population.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

NOTICE

The value of this parameter must be at least **2**, and useful values are typically from **100** to **1000**. If this parameter is set to **0**, DWS selects a proper value based on **geqo_effort** and the number of tables.

Default value: 0

geqo_generations

Parameter description: Specifies the number parameter iterations of the algorithm used by GEQO.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

NOTICE

The value of this parameter must be at least **1**, and useful values are typically from **100** to **1000**. If it is set to **0**, a suitable value is chosen based on **geqo_pool_size**.

Default value: 0

geqo_selection_bias

Parameter description: Specifies the selection bias used by GEQO. The selection bias is the selective pressure within the population.

Type: USERSET

Value range: a floating point number ranging from 1.5 to 2.0

Default value: 2

NOTE

A larger value indicates higher selective pressure, and the probability of selecting better individuals is higher. A smaller value indicates lower selection pressure and more diversity.

geqo_seed

Parameter description: Specifies the initial value of the random number generator used by GEQO to select random paths through the join order search space.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Varying the value changes the setting of join paths explored, and may result in a better or worse path being found.

Default value: 0

19.9.4 Other Optimizer Options

default_statistics_target

Parameter description: Specifies the default statistics target for table columns without a column-specific target set via **ALTER TABLE SET STATISTICS**. If this parameter is set to a positive number, it indicates the number of samples of statistics information. If this parameter is set to a negative number, percentage is used to set the statistic target. The negative number converts to its corresponding percentage, for example, -5 means 5%. During sampling, a random sample size is determined by multiplying the **default_statistics_target** by 300. For example, if the default value is **100**, then 30,000 pages will be randomly read and 30,000 data records will be randomly selected from them to complete the random sampling.

Type: USERSET

Value range: an integer ranging from -100 to 10000

NOTICE

- A larger positive number than the parameter value increases the time required to do **ANALYZE**, but might improve the quality of the optimizer's estimates.
- Changing settings of this parameter may result in performance deterioration. If query performance deteriorates, you can:
 1. Restore to the default statistics.
 2. Use hints to optimize the query plan. For details, see [Hint-based Tuning](#).
- If this parameter is set to a negative value, the number of samples is greater than or equal to 2% of the total data volume, and the number of records in user tables is less than 1.6 million, the time taken by running **ANALYZE** will be longer than when this parameter uses its default value.
- **AUTOANALYZE** does not allow you to set a sampling size for temporary table sampling. Its default value will be used for sampling.
- If statistics are forcibly calculated based on memory, the sampling size is limited by the **maintenance_work_mem** parameter.

Default value: 100

random_function_version

Parameter description: Specifies the random function version selected by ANALYZE during data sampling. This feature is supported only in 8.1.2 or later.

Type: USERSET

Value range: enumerated values

- The value **0** indicates that the random function provided by the C standard library is used.
- The value **1** indicates that the optimized and enhanced random function is used.

Default value:

- If the current cluster is upgraded from an earlier version to 8.2.0.100, the default value is **0** to ensure forward compatibility.
- If the cluster version 8.2.0.100 is newly installed, the default value is **1**.

constraint_exclusion

Parameter description: Controls the query optimizer's use of table constraints to optimize queries.

Type: USERSET

Value range: enumerated values

- **on** indicates the constraints for all tables are examined.
- **off**: No constraints are examined.
- **partition** indicates that only constraints for inherited child tables and **UNION ALL** subqueries are examined.

NOTICE

When **constraint_exclusion** is set to **on**, the optimizer compares query conditions with the table's **CHECK** constraints, and omits scanning tables for which the conditions contradict the constraints.

Default value: partition

 NOTE

Currently, this parameter is set to **on** by default to partition tables. If this parameter is set to **on**, extra planning is imposed on simple queries, which has no benefits. If you have no partitioned tables, set it to **off**.

cursor_tuple_fraction

Parameter description: Specifies the optimizer's estimated fraction of a cursor's rows that are retrieved.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

NOTICE

Smaller values than the default value bias the optimizer towards using **fast start** plans for cursors, which will retrieve the first few rows quickly while perhaps taking a long time to fetch all rows. Larger values put more emphasis on the total estimated time. At the maximum setting of **1.0**, cursors are planned exactly like regular queries, considering only the total estimated time and how soon the first rows might be delivered.

Default value: 0.1

from_collapse_limit

Parameter description: Specifies whether the optimizer merges sub-queries into upper queries based on the resulting FROM list. The optimizer merges sub-queries into upper queries if the resulting FROM list would have no more than this many items.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

Default value: 8

 CAUTION

Smaller values reduce planning time but may lead to inferior execution plans.

join_collapse_limit

Parameter description: Specifies whether the optimizer rewrites **JOIN** constructs (except **FULL JOIN**) into lists of **FROM** items based on the number of the items in the result list.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

Default value: 8

NOTICE

- Setting this parameter to **1** prevents join reordering. As a result, the join order specified in the query will be the actual order in which the relations are joined. The query optimizer does not always choose the optimal join order. Therefore, advanced users can temporarily set this variable to **1**, and then specify the join order they desire explicitly.
 - Smaller values reduce planning time but lead to inferior execution plans.
-

join_search_mode

Parameter description: plan path search mode.

Type: USERSET

Value range: enumerated values

- **exhaustive:** Traditional dynamic planning and genetic methods are used to search for planned paths.
- **heuristic:** The heuristic method is used to search for planned paths. This method improves the plan generation performance, but there is a possibility that the optimal plan is ignored. This setting only takes effect for scenarios where a Drive Hint is specified or the number of joined tables exceeds **from_collapse_limit**.

Default value: **heuristic**

enable_from_collapse_hint

Parameter description: Specifies whether to rewrite the **FROM** list to make the hint take effect, and then rewrite it again based on the **from_collapse_limit** and **join_collapse_limit** parameters. This parameter is supported by clusters of version 8.2.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the **FROM** list is first rewritten in hint mode.
- **off** indicates that the **FROM** list is rewritten without difference.

NOTICE

- If this parameter is enabled, the optimizer preferentially rewrites the **FROM** list in hint mode. However, you can learn whether a hint takes effect only after the plan is generated.
- If this parameter is disabled, the plan is generated in the same way as that in versions earlier than 8.2.0. That is, the plan is generated regardless of whether the table has hints.

Default value: on

plan_mode_seed

Parameter description: This is a commissioning parameter. Currently, it supports only OPTIMIZE_PLAN and RANDOM_PLAN. **OPTIMIZE_PLAN** indicates the optimal plan, the cost of which is estimated using the dynamic planning algorithm, and its value is 0. **RANDOM_PLAN** indicates the plan that is randomly generated. If **plan_mode_seed** is set to -1, you do not need to specify the value of the seed identifier. Instead, the optimizer generates a random integer ranging from 1 to 2147483647, and then generates a random execution plan based on this random number. If **plan_mode_seed** is set to an integer ranging from 1 to 2147483647, you need to specify the value of the seed identifier, and the optimizer generates a random execution plan based on the seed value.

Type: USERSET

Value range: an integer ranging from -1 to 2147483647

Default value: 0

NOTICE

- If **plan_mode_seed** is set to **RANDOM_PLAN**, the optimizer generates different random execution plans, which may not be the optimal. Therefore, to guarantee the query performance, the default value 0 is recommended during upgrade, scale-out, scale-in, and O&M.
- If this parameter is not set to 0, the specified hint will not be used.

enable_hdfs_predicate_pushdown

Parameter description: Specifies whether the function of pushing down predicates the native data layer is enabled.

Type: SUSET

Value range: Boolean

- **on** indicates this function is enabled.
- **off** indicates this function is disabled.

Default value: on

windowagg_pushdown_enhancement

Parameter description: Specifies whether to enable enhanced predicate pushdown for window functions in aggregation scenarios. (This parameter is supported only by clusters of version 8.2.0 or later.)

Type: SUSET

Value range: Boolean

- **on** indicates that the predicate pushdown enhancement for window functions is enabled in aggregation scenarios.
- **off** indicates that the predicate pushdown enhancement for window functions is disabled in aggregation scenarios.

Default value: on

implied_quality_optmode

Parameter description: Specifies how to pass conditions for the equivalent columns in a statement. (This parameter is supported only by clusters of version 8.2.0 or later.)

Type: SUSET

Value range: enumerated values

- **normal** indicates forward compatibility with 8.1.3 and earlier versions, that is, the implied expression behavior is optimized.
- **negative** indicates that the implied expression behavior is not optimized.
- **positive** indicates that type conversion expressions are optimized in addition to the operations specified by **normal**.

Default value: normal

enable_random_datanode

Parameter description: Specifies whether the function that random query about DN in the replication table is enabled. A complete data table is stored on each DN for random retrieval to release the pressure on nodes.

Type: USERSET

Value range: Boolean

- **on:** This function is enabled.
- **off:** This function is disabled.

Default value: on

hashagg_table_size

Parameter description: Specifies the hash table size during **HASH AGG** execution.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX/2

Default value: 0

enable_codegen

Parameter description: Specifies whether code optimization can be enabled. Currently, the code optimization uses the LLVM optimization.

Type: USERSET

Value range: Boolean

- **on** indicates code optimization can be enabled.
- **off** indicates code optimization cannot be enabled.

Default value: off

CAUTION

- In clusters of version 9.1.0.218 or later (new installation and upgrade scenarios), the default value of this parameter is **off**. In clusters of version earlier than 9.1.0.218, the default value of this parameter is **on**.
 - Currently, the LLVM optimization only supports the vectorized executor and SQL on Hadoop features. You are advised to set this parameter to **off** in other cases.
-

codegen_strategy

Parameter description: Specifies the codegen optimization strategy that is used when an expression is converted to codegen-based.

Type: USERSET

Value range: enumerated values

- **partial** indicates that you can still call the LLVM dynamic optimization strategy using the codegen framework of an expression even if functions that are not codegen-based exist in the expression.
- **pure** indicates that the LLVM dynamic optimization strategy can be called only when all functions in an expression can be codegen-based.

NOTICE

In the scenario where query performance reduces after the codegen function is enabled, you can set this parameter to **pure**. In other scenarios, do not change the default value **partial** of this parameter.

Default value: partial

enable_codegen_print

Parameter description: Specifies whether the LLVM IR function can be printed in logs.

Type: USERSET

Value range: Boolean

- **on** indicates that the LLVM IR function can be printed in logs.
- **off** indicates that the LLVM IR function cannot be printed in logs.

Default value: off

codegen_cost_threshold

Parameter description: The LLVM compilation takes some time to generate executable machine code. Therefore, LLVM compilation is beneficial only when the actual execution cost is more than the sum of the code required for generating machine code and the optimized execution cost. This parameter specifies a threshold. If the estimated execution cost exceeds the threshold, LLVM optimization is performed.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 10000

llvm_compile_expr_limit

Parameter description: Specifies the limit for compiling expressions with LLVM. If there are more expressions than the limit, only the first ones are compiled and an alarm is generated. (To allow the alarm to be generated, execute **SET analysis_options="on(LLVM_COMPILE)"** before **explain performance** is executed.)

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX

Default value: 500

llvm_compile_time_limit

Parameter description: If the percentage of the LLVM compilation time to the executor running time exceeds the threshold specified by **llvm_compile_time_limit**, an alarm is generated. (To allow the alarm to be generated, execute **SET analysis_options="on(LLVM_COMPILE)"** before **explain performance** is executed.) This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

Default value: 0.2

enable_constraint_optimization

Parameter description: Specifies whether the informational constraint optimization execution plan can be used for an HDFS foreign table.

Type: SUSET

Value range: Boolean

- **on** indicates the plan can be used.
- **off** indicates the plan cannot be used.

Default value: on

enable_bloom_filter

Parameter description: Specifies whether the BloomFilter optimization is used.

Type: USERSET

Value range: Boolean

- **on** indicates the BloomFilter optimization can be used.
- **off** indicates the BloomFilter optimization cannot be used.

Default value: on

NOTICE

Scenario: If in a HASH JOIN, the thread of the foreign table contains HDFS tables or column-store tables, the Bloom filter is triggered.

Constraints:

1. Only **INNER JOIN**, **SEMI JOIN**, **RIGHT JOIN**, **RIGHT SEMI JOIN**, **RIGHT ANTI JOIN** and **RIGHT ANTI FULL JOIN** are supported.
 2. JOIN condition of the internal table: It cannot be an expression for HDFS internal or foreign tables. It can be an expression for column-store tables, but only at the non-join layer.
 3. The join condition of the foreign table must be simple column join.
 4. When the join conditions of the internal and foreign tables (HDFS) are both simple column joins, the estimated data that can be removed at the plan layer must be over 1/3.
 5. Joined columns cannot contain NULL values.
 6. Data type:
 - HDFS internal and foreign tables support SMALLINT, INTEGER, BIGINT, REAL/FLOAT4, DOUBLE PRECISION/FLOAT8, CHAR(n)/CHARACTER(n)/NCHAR(n), VARCHAR(n)/CHARACTER VARYING(n), CLOB and TEXT.
 - Column-store tables support SMALLINT, INTEGER, BIGINT, OID, "char", CHAR(n)/CHARACTER(n)/NCHAR(n), VARCHAR(n)/CHARACTER VARYING(n), NVARCHAR2(n), CLOB, TEXT, DATE, TIME, TIMESTAMP and TIMESTAMPTZ. The collation of the character type must be C.
-

runtime_filter_type

Parameter description : Specifies the type of runtime filter used, and only takes effect when [enable_bloom_filter](#) is enabled. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: enumerated values

- **All** indicates that the runtime filters in all scenarios are used.
- **Topn_filter** indicates the runtime filters in the **ORDER BY** scenario with **LIMIT** are used.
- **Bloom_filter** indicates that only runtime filters in join scenarios are used, and a bloom filter is generated for filtering after meeting certain conditions.
- **Min_max** indicates that only the runtime filters in join scenarios are used and only a **min_max** filter is generated for filtering.
- **None** indicates that no runtime filters are used, and only the original bloom filter has filtering effect.

Default value: All

NOTICE

- Application scenario: Plan type of the **HASH JOIN** foreign table in a column-store table and the **ORDER BY** plan type with **LIMIT** in a column-store table.
 - Constraints:
 - The usage restrictions for JOIN scenarios are the same as those for the [enable_bloom_filter](#) parameter.
 - In the order by scenario with limit, the order by field types only support **SMALLINT**, **INTEGER**, **BIGINT**, **"char"**, **CHAR(n)/CHARACTER(n)/NCHAR(n)**, **VARCHAR(n)/CHARACTER VARYING(n)**, **NVARCHAR2(n)**, **TEXT**, **DATE**, **TIME**, **TIMESTAMP**, and **TIMESTAMPZ**, and the sorting rules for character types must be specified as **C**.
-

runtime_filter_ratio

Parameter description: Specifies the threshold for using bloom filter for fine-grained row-level filtering in join scenarios in runtime filter, and only takes effect when [runtime_filter_type](#) is set to a value greater than or equal to **Bloom_filter**. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: a floating point number ranging from 0.0 to 1.0

Default value: 0.01

NOTICE

- Application scenario: HASH JOIN of column-store tables, where the internal table **estimate_join_rows**/foreign table **estimate_join_rows** ≤ **runtime_filter_ratio**. Fine-grained row-level filtering is only recommended for join scenarios where there is a significant difference in data volume between the internal and foreign tables. Improper **runtime_filter_ratio** settings may lead to degraded performance in join scenarios.
- Usage restrictions: Fine-grained row-level filtering is only supported for join field types of **SMALLINT**, **INTEGER**, **BIGINT**, and **FLOAT**.

runtime_filter_cost_options

Parameter description : Specifies whether to generate a runtime filter plan based on the cost. This is supported only by clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: a string

- **apply_partial**: The runtime filter path can be generated as long as the **build** end contains a table required by a runtime filter on the **probe** end.
- **apply_all**: The runtime filter path can be generated only when the **build** end contains all tables required by the runtime filter that can be applied to the **probe** end.

Default value: "", indicating that runtime filters are not used during plan generation, regardless of the cost.

NOTICE

If both **apply_partial** and **apply_all** are set, the setting of **apply_all** takes effect.

enable_extrapolation_stats

Parameter description: Specifies whether to use the extrapolation logic based on historical statistics. Using this logic may increase the accuracy of estimation for tables whose statistics have not been collected. However, there is also a possibility that the estimation is too large due to incorrect inference.

Type: USERSET

Value range: Boolean

- **on** indicates that the extrapolation logic is used for data of DATE type based on historical statistics.
- **off** indicates that the extrapolation logic is not used for data of DATE type based on historical statistics.

Default value:

- If the current cluster is upgraded from an earlier version to 8.2.0.100, the default value is **off** to ensure forward compatibility.

- If the cluster version 8.2.0.100 is newly installed, the default value is **on**.

autoanalyze

Parameter description: Specifies whether to allow automatic statistics collection for a table that has no statistics or a table whose amount of data modification reaches the threshold for triggering **ANALYZE** when a plan is generated. In this case, **AUTOANALYZE** cannot be triggered for foreign tables or temporary tables with the **ON COMMIT [DELETE ROWS|DROP]** option. To collect statistics, you need to manually perform the **ANALYZE** operation. If an exception occurs in the database during the execution of autoanalyze on a table, after the database is recovered, the system may still prompt you to collect the statistics of the table when you run the statement again. In this case, manually perform the **ANALYZE** operation on the table to synchronize statistics.

NOTICE

If the amount of data modification reaches the threshold for triggering **ANALYZE**, the amount of data modification exceeds **autovacuum_analyze_threshold + autovacuum_analyze_scale_factor * reltuples**. *reltuples* indicates the estimated number of rows in the table recorded in **pg_class**.

Type: SUSET

Value range: Boolean

- **on** indicates that the table statistics are automatically collected.
- **off** indicates that the table statistics are not automatically collected.

Default value: on

enable_analyze_partition

Parameter description: Specifies whether to support collecting statistics for a specific partition of a table. After enabling this parameter, you can collect statistics for a specific partition using **ANALYZE table_name PARTITION (partition_name)**, and when querying data on this partition of the table, the optimizer will choose to use partition statistics.

Type: USERSET

Value range: Boolean

- **on** indicates supporting collecting statistics for a specific partition of a table.
- **off** indicates that collecting statistics for a specific partition of a table is not supported.

Default value: off

analyze_use_dn_correlation

Parameter description: Specifies whether CNs use correlation statistics of DN's when executing **ANALYZE**. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that CNs use correlation statistics of DNs.
- **off** indicates that CNs do not use correlation statistics of DNs.

Default value: on

analyze_predicate_column_threshold

Parameter description: Specifies whether to enable ANALYZE operations for predicate columns and the minimum number of columns supported. This is supported only by clusters of version 9.1.0.100 or later.

Type: SIGHUP

Value range: an integer ranging from 0 to 10000

- The value **0** indicates that ANALYZE operations are disabled for predicate columns and predicate columns are not collected or analyzed.
- A value greater than 0 indicates that predicate column collection is enabled and predicate column analysis is performed only on tables whose number of columns is greater than or equal to the value of this parameter.

Default value: 10

enable_runtime_analyze_concurrent

Parameter description: Specifies whether to support concurrent RUNTIME ANALYZE operations on a table. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that concurrent operations are supported.
- **off** indicates that concurrent operations are not supported.

Default value: on

analyze_max_columns_count

Parameter description: Specifies the maximum number of columns supported by ANALYZE. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: an integer ranging from -1 to 10000

- **-1** indicates that the number of columns supported by ANALYZE is not limited.
- A value greater than **-1** indicates that only columns up to this value will be collected, and any columns beyond this value will not be collected.

Default value: -1

query_dop

Parameter description: Specifies the user-defined degree of parallelism.

Type: USERSET

Value range: an integer ranging from -64 to 64.

[1, 64]: Fixed SMP is enabled, and the system will use the specified degree.

0: SMP adaptation function is enabled. The system dynamically selects the optimal parallelism degree [1,8] (x86 platforms) or [1,64] (Kunpeng platforms) for each query based on the resource usage and query plans.

[-64, -1]: SMP adaptation is enabled, and the system will dynamically select a degree from the limited range.

NOTE

- For TP services that mainly involve short queries, if services cannot be optimized through lightweight CNs or statement delivery, it will take a long time to generate an SMP plan. You are advised to set **query_dop** to **1**. For AP services with complex statements, you are advised to set **query_dop** to **0**.
- After enabling concurrent queries, ensure you have sufficient CPU, memory, network, and I/O resources to achieve the optimal performance.
- To prevent performance deterioration caused by an overly large value of **query_dop**, the system calculates the maximum number of available CPU cores for a DN and uses the number as the upper limit for this parameter. If the value of **query_dop** is greater than 4 and also the upper limit, the system resets **query_dop** to the upper limit.

Default value: 1 (0 for cloud flavors with 64 GB or larger memory)

query_dop_ratio

Parameter description: Specifies the DOP multiple used to adjust the optimal DOP preset in the system when **query_dop** is set to **0**. That is, $DOP = \text{Preset DOP} \times \text{query_dop_ratio}$ (ranging from 1 to 64). If this parameter is set to **1**, the DOP cannot be adjusted.

Type: USERSET

Value range: a floating point number ranging from 0 to 64

Default value: 1

debug_group_dop

Parameter description: Specifies the unified DOP parallelism degree allocated to the groups that use the Stream operator as the vertex in the generated execution plan when the value of **query_dop** is **0**. This parameter is used to manually specify the DOP for specific groups for performance optimization. Its format is **G1,D1,G2,D2,...**, where **G1** and **G2** indicate the group IDs that can be obtained from logs and **D1** and **D2** indicate the specified DOP values and can be any positive integers.

Type: USERSET

Value range: a string

Default value: empty

NOTICE

This parameter is used only for internal optimization and cannot be set. You are advised to use the default value.

enable_analyze_check

Parameter description: Checks whether statistics were collected about tables whose **reltuples** and **relpages** are shown as **0** in **pg_class** during plan generation. **This parameter has been discarded in clusters of version 8.1.3 or later, but is reserved for compatibility with earlier versions. The setting of this parameter does not take effect.**

Type: SUSET

Value range: Boolean

- **on** enables the check.
- **off** disables the check.

Default value: on

enable_sonic_hashagg

Parameter description: Specifies whether to use the Hash Agg operator for column-oriented hash table design when certain constraints are met.

Type: USERSET

Value range: Boolean

- **on** indicates that the Hash Agg operator is used for column-oriented hash table design when certain constraints are met.
- **off** indicates that the Hash Agg operator is not used for column-oriented hash table design.

 NOTE

- If **enable_sonic_hashagg** is enabled and certain constraints are met, the Hash Agg operator will be used for column-oriented hash table design, and the memory usage of the operator can be reduced. However, in scenarios where the code generation technology (enabled by [enable_codegen](#)) can significantly improve performance, the performance of the operator may deteriorate.
- If **enable_sonic_hashagg** is set to **on**, when certain constraints are met, the hash aggregation operator designed for column-oriented hash tables is used and its name is displayed as **Sonic Hash Aggregation** in the output of the Explain Analyze/Performance operation. When the constraints are not met, the operator name is displayed as **Hash Aggregation**.

Default value: on

enable_sonic_hashjoin

Parameter description: Specifies whether to use the Hash Join operator for column-oriented hash table design when certain constraints are met.

Type: USERSET

Value range: Boolean

- **on** indicates that the Hash Join operator is used for column-oriented hash table design when certain constraints are met.
- **off** indicates that the Hash Join operator is not used for column-oriented hash table design.

NOTE

- Currently, the parameter can be used only for Inner Join.
- If **enable_sonic_hashjoin** is enabled, the memory usage of the Hash Inner operator can be reduced. However, in scenarios where the code generation technology can significantly improve performance, the performance of the operator may deteriorate.
- If **enable_sonic_hashjoin** is set to **on**, when certain constraints are met, the hash join operator designed for column-oriented hash tables is used and its name is displayed as **Sonic Hash Join** in the output of the Explain Analyze/Performance operation. When the constraints are not met, the operator name is displayed as **Hash Join**.

Default value: on

enable_sonic_optspill

Parameter description: Specifies whether to optimize the number of hash join or hash agg files spilled to disks in the sonic scenario. This parameter takes effect only when **enable_sonic_hashjoin** or **enable_sonic_hashagg** is enabled.

Type: USERSET

Value range: Boolean

- **on** indicates that the number of files spilled to disks is optimized.
- **off** indicates that the number of files spilled to disks is not optimized.

NOTE

For the hash join or hash agg operator that meets the sonic criteria, if this parameter is set to **off**, one file is spilled to disks for each column. If this parameter is set to **on** and the data types of different columns are similar, only one file (a maximum of five files) will be spilled to disks.

Default value: on

expand_hashtable_ratio

Parameter description: Specifies the expansion ratio used to resize the hash table during the execution of the Hash Agg and Hash Join operators.

Type: USERSET

Value range: a floating point number of 0 or ranging from 0.5 to 10

 NOTE

- Value **0** indicates that the hash table is adaptively expanded based on the current memory size.
- The value ranging from 0.5 to 10 indicates the multiple used to expand the hash table. Generally, a larger hash table delivers better performance but occupies more memory space. If the memory space is insufficient, data may be spilled to disks in advance, causing performance deterioration.

Default value: 0

plan_cache_mode

Parameter description: Specifies the policy for generating an execution plan in the **prepare** statement.

Type: USERSET

Value range: enumerated values

- **auto** indicates that the **custom plan** or **generic plan** is selected by default.
- **force_generic_plan** indicates that the **generic plan** is forcibly used.
- **force_custom_plan** indicates that the **custom plan** is forcibly used.

 NOTE

- This parameter is valid only for the **prepare** statement. It is used when the parameterized field in the **prepare** statement has severe data skew.
- The **custom plan** is a plan generated after you run a prepared statement where parameters in the EXECUTE statement are embedded. The **custom plan** generates a plan based on specific parameters in the execute statement. This scheme generates a preferred plan based on specific parameters each time and has good execution performance. The disadvantage is that the plan needs to be regenerated before each execution, resulting in a large amount of repeated optimizer overhead.
- The **generic plan** is a plan generated after you run a prepared statement. The plan policy binds parameters to the plan when you run the EXECUTE statement and execute the plan. The advantage of this solution is that repeated optimizer overheads can be avoided in each execution. The disadvantage is that the plan may not be optimal when data skew occurs for the bound parameter field. When some bound parameters are used, the plan execution performance is poor.

Default value: auto

wlm_query_accelerate

Parameter description: Specifies whether the query needs to be accelerated when short query acceleration is enabled.

Type: USERSET

Value range: an integer ranging from -1 to 1

- **-1:** indicates that short queries are controlled by the fast lane, and the long queries are controlled by the slow lane.
- **0:** indicates that queries are not accelerated. Both short and long queries are controlled by the slow lane.
- **1:** indicates that queries are accelerated. Both short queries and long queries are controlled by the fast lane.

Default value: -1

show_unshippable_warning

Parameter description: Specifies whether to print the alarm for the statement pushdown failure to the client.

Type: USERSET

Value range: Boolean

- **on:** Records the reason why the statement cannot be pushed down in a WARNING log and prints the log to the client.
- **off:** Logs the reason why the statement cannot be pushed down only.

Default value: off

hashjoin_spill_strategy

Parameter description: specifies the hash join policy for spilling data to disks. This feature is supported in 8.1.2 or later.

Type: USERSET

Value range: The value is an integer ranging from 0 to 6.

- **0:** If an inner table is too large to be fully stored in database memory, the table will be partitioned. If the table cannot be further partitioned and there is not enough memory for storing it, the system will check whether the foreign table can be stored in memory and be used to create a hash table. If the foreign table can be stored in the memory and used to create a hash table, HashJoin will be performed. Otherwise, NestLoop will be performed.
- **1:** If an inner table is too large to be fully stored in database memory, the table will be partitioned. If the table cannot be further partitioned and there is still not enough memory for storing it, the system will check whether the foreign table can be stored in memory and be used to create a hash table. If both the inner and outer tables are large, a hash join is forcibly performed.
- **2:** If the size of the inner table is large and cannot be partitioned after data is spilled to disks for multiple times, HashJoin will be forcibly performed.
- **3:** If the size of the inner table is large and cannot be partitioned after data is spilled to disks for multiple times, the system attempts to place the outer table in the available memory of the database to create a hash table. If both the inner and outer tables are large, an error is reported.
- **4:** If the size of the inner table is large and cannot be partitioned after data is spilled to disks for multiple times, an error is reported.
- **5:** If the inner table is large and cannot be fully stored in database memory, and the foreign table can be fully stored in memory, the foreign table will be used to create a hash table and perform HashJoin. If the foreign table cannot be fully stored in memory, it will be partitioned until the inner and foreign tables cannot be further partitioned. Then, NestLoop will be performed.
- **6:** If the inner table is large and cannot be fully stored in database memory, and the foreign table can be fully stored in memory, the foreign table will be used to create a hash table and perform HashJoin. If the foreign table cannot be fully stored in memory, it will be partitioned until the inner and foreign

tables cannot be further partitioned. Then, HashJoin will be forcibly performed.

 NOTE

- This parameter is valid only for a vectorized hash join operator.
- If the number of distinct values is small and the data volume is large, data may fail to be flushed to disks. As a result, the memory usage is too high and the memory is out of control. If this parameter is set to **0**, the system attempts to swap the inner and outer tables or perform a nested loop join to prevent this problem. However, a nested loop join may deteriorate performance in some scenarios. In this case, this parameter can be set to **1**, **2**, or **6** to forcibly perform HashJoin.
- The value **0** does not take effect for a vectorized full join, and the behavior is the same as that of the value **1**. The system attempts to create a hash table only for the outer table and does not perform a nested loop join.
- If the inner table is too large to be fully stored in memory, but the foreign table can be stored in memory, you are advised to set this parameter to **5** or **6** rather than **0** or **1**, directly performing Hashjoin on the foreign table without multiple rounds of partitioning and spill to disk. If a foreign table contains only a small amount of distinct data, creating a hash table using the foreign table may cause performance deterioration. In this case, you can change the value of this parameter to **0** or **1**.

Default value: 0

max_streams_per_query

Parameter description: Controls the number of Stream nodes in a query plan. (This parameter is supported only in 8.1.1 and later cluster versions.)

Type: SUSET

Value range: an integer ranging from -1 to 10000.

- **-1** indicates that the number of Stream nodes in the query plan is not limited.
- A value within the range **0** to **10000** indicates that when the number of Stream nodes in the query plan exceeds the specified value, an error is reported and the query plan will not be executed.

 NOTE

- This parameter controls only the Stream nodes on DN and does not control the Gather nodes on the CN.
- This parameter does not affect the EXPLAIN query plan, but affects EXPLAIN ANALYZE and EXPLAIN PERFORMANCE.

Default value: -1

enable_agg_limit_opt

Parameter description: Specifies whether to optimize **select distinct col from table limit N**. This parameter is valid only if N is less than 16,384. The parameter **table** indicates a column-store table. This parameter is supported only by clusters of version 8.2.0.101 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimization is enabled. After this function is enabled, query results are from different DN, and you do not need to create a full hash table on each DN, significantly improving query performance.
- **off** indicates that the optimization is disabled.

Default value: on

stream_ctescan_pred_threshold

Parameter description: minimum number of filter criteria contained in a CTE when **enable_stream_ctescan** is set to **on** and the CTE contains only a single table filtering condition. If the value is greater than or equal to the value of this parameter, the share scan mode is used. If the value is less than the value of this parameter, the inline mode is used. This parameter is supported only by clusters of version 8.2.1 or later.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 2

stream_ctescan_max_estimate_mem

Parameter description: maximum estimated memory value of the CTE when **enable_stream_ctescan** is set to **on**. This parameter must be used together with **stream_ctescan_refcount_threshold**. If the estimated memory is greater than the value of **stream_ctescan_max_estimate_mem** and the number of references is less than the value of **stream_ctescan_refcount_threshold**, the inline mode is used. Otherwise, the sharescan mode is used. This parameter is supported only by clusters of version 8.2.1 or later.

Type: SUSET

Value range: an integer ranging from 32 x 1024 (32 MB) to INT_MAX, in KB.

Default value: 256 MB

stream_ctescan_refcount_threshold

Parameter description: maximum number of times that the CTE can be referenced when **enable_stream_ctescan** is set to **on**. This parameter must be used together with **stream_ctescan_max_estimate_mem**. If the estimated memory is greater than the value of **stream_ctescan_max_estimate_mem** and the number of references is less than the value of **stream_ctescan_refcount_threshold**, the inline mode is used. Otherwise, the sharescan mode is used. This parameter is supported only by clusters of version 8.2.1 or later.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 4

 NOTE

This parameter takes effect only when the value is greater than 0. When the value is 0, only **stream_ctescan_max_estimate_mem** is used to control the inline behavior.

inlist_rough_check_threshold

Parameter description: Specifies the maximum number of values in the **IN** condition when **enable_csqual_pushdown** is enabled and the filter criterion is **IN** for rough check pushdown. If the number of values in the **IN** filter condition exceeds the value of this parameter, the maximum and minimum values in the **IN** filter condition are used for pushdown. This parameter is supported only by clusters of version 8.2.0.101 or later.

Type: SUSET

Value range: an integer ranging from 0 to 10000

Default value: 100

 NOTE

If the **IN** condition is executed on the only distribution column of a table, values can be filtered on DN. In this case, the maximum number of values in the **IN** condition is **inlist_rough_check_threshold** multiplied by the number of DN.

enable_array_optimization

Parameter description: whether to split the Array type generated by the **IN**, **ANY**, or **ALL** condition into common expressions for execution. This parameter will support multiple optimizations such as vectorized execution, rough check pruning, and partition pruning. This parameter is supported only by clusters of version 8.2.1 or later.

Type: SUSET

Value range: Boolean

- **on** indicates that expressions of the Array type are split for optimization.
- **off** indicates that expressions of the Array type are not split for optimization.

Default value: on

max_skew_num

Parameter description: controls the number of skew values allowed by the optimizer for redistribution optimization. This parameter is supported only by clusters of version 8.2.1 or later.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 10

enable_dict_plan

Parameter description: Specifies whether the optimizer uses dictionary encoding to speed up queries that use operators such as **Group By** and **Filter**. This parameter is supported only by clusters of 8.3.0 or later.

Type: USERSET

Value range: Boolean

- **on:** enables the optimizer dictionary encoding.
- **off:** disables the optimizer dictionary encoding.

Default value: **off**

dict_plan_distinct_limit

Parameter description: Specifies the distinct value of a column in a table. Dictionary encoding is enabled only when the value is less than or equal to the threshold. This parameter is supported only by clusters of 8.3.0 or later.

Type: USERSET

Value range: 0 to INT_MAX

Default value: 10000

NOTE

The two parameters **dict_plan_distinct_limit** and **dict_plan_duplicate_ratio** determine if dictionary encoding is applied.

dict_plan_duplicate_ratio

Parameter description: Specifies the repetition rate threshold of a column. Dictionary encoding is enabled only when the repetition rate of the column is greater than or equal to the threshold. Dictionary encoding is suitable for columns with a small number of distinct values and a high repetition rate. This parameter is supported only by clusters of 8.3.0 or later.

Type: USERSET

Value range: 0.0 to 100, in percentage

Default value: 90

NOTE

The two parameters **dict_plan_distinct_limit** and **dict_plan_duplicate_ratio** determine if dictionary encoding is applied.

enable_cu_predicate_pushdown

Parameter description:

1. Function overview: Specifies whether simple filter criteria are pushed down to the CU for filtering. Enabling this will enhance query performance, particularly when working with the **bitmap_columns** column and PCK sorting column. It

applies to specific **WHERE**, **IS NULL**, and **IN** conditions. This parameter is supported only by clusters of 8.3.0 or later.

2. Supported column types:

- Integer type: INT2, INT4, and INT8
- Date and time type: DATE, TIMESTAMP, and TIMESTAMPTZ
- String types: VARCHAR and TEXT
- Numeral type: NUMERIC (a maximum of 19 characters)

3. Query conditions: This function supports multiple **WHERE** expressions, including:

- **IN** expression: matches multiple values.
- **IS NULL** / **IS NOT NULL** condition: checks whether the column value is null.
- Comparison expressions: greater than (>), less than (<), equal to (=), and not equal to (<>), which is used for range query and exact match.

Type: USERSET

Value range: Boolean

- **on:** Simple filter criteria are pushed down to the CU for filtering.
- **off:** Simple filter criteria are not pushed down to the CU for filtering.

Default value: on

 NOTE

The basic filter conditions for the dictionary column include the equality operator (=), the **IN** expression, and the **IS (NOT) NULL** condition. This filter is known as the CU Predicate Filter, as it is pushed down to the storage layer and applied during the VectorBatch filling process.

info_constraint_options

Parameter description: Specifies whether or which kind of informational constraints can be created. This is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: enumerated values

- **none:** indicates that no informational constraint can be created.
- **foreign_key:** indicates that foreign key constraints can be created.

Default value: none

19.10 Error Reporting and Logging

19.10.1 Logging Destination

log_statement_length_limit

Parameter description: Specifies the length of SQL statements to be printed. If the length of an SQL statement exceeds the specified value, the SQL statement is recorded in the **statement-*Timestamp*.log** file. This parameter is supported only by 9.1.0.200 and later versions.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX.

Default value: 1024

19.10.2 Logging Time

client_min_messages

Parameter description: Specifies which level of messages are sent to the client. Each level covers all the levels following it. The lower the level is, the fewer messages are sent.

Type: USERSET

NOTICE

When the values of **client_min_messages** and **log_min_messages** are the same, the levels are different.

Valid values: Enumerated values. Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error** For details about the parameters, see [Table 19-1](#).

Default value: notice

log_min_messages

Parameter description: Specifies which level of messages will be written into server logs. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

Type: SUSET

NOTICE

When the values of **client_min_messages** and **log_min_messages** are the same, the levels are different.

Value range: enumerated type. Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error, fatal, panic** For details about the parameters, see [Table 19-1](#).

Default value: warning

log_min_error_statement

Parameter description: Specifies which SQL statements that cause errors condition will be recorded in the server log.

Type: SUSET

Value range: enumerated type. Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error, fatal, panic** For details about the parameters, see [Table 19-1](#).

NOTE

- The default is **error**, indicating that statements causing errors, log messages, fatal errors, or panics will be logged.
- **panic**: This feature is disabled.

Default value: error

log_min_duration_statement

Parameter description: Specifies the threshold for logging statement execution durations. The execution duration that is greater than the specified value will be logged.

This parameter helps track query statements that need to be optimized. For clients using extended query protocol, durations of the Parse, Bind, and Execute are logged independently.

Type: SUSET

NOTICE

If this parameter and [log_statement](#) are used at the same time, statements recorded based on the value of **log_statement** will not be logged again after their execution duration exceeds the value of this parameter. If you are not using **syslog**, it is recommended that you log the process ID (PID) or session ID using **log_line_prefix** so that you can link the current statement message to the last logged duration.

Value range: an integer ranging from -1 to INT_MAX. The unit is millisecond.

- If this parameter is set to **250**, execution durations of SQL statements that run 250 ms or longer will be logged.
- **0**: Execution durations of all the statements are logged.
- **-1**: This feature is disabled.

Default value: 30min

backtrace_min_messages

Parameter description: Prints the function's stack information to the server's log file if the level of information generated is greater than or equal to this parameter level.

Type: SUSET

NOTICE

This parameter is used for locating customer on-site problems. Because frequent stack printing will affect the system's overhead and stability, therefore, when you locate the onsite problems, set the value of this parameter to ranks other than **fatal** and **panic**.

Value range: enumerated values

Valid values: **debug5, debug4, debug3, debug2, debug1, info, log, notice, warning, error, fatal, panic** For details about the parameters, see [Table 19-1](#).

Default value: panic

[Table 19-1](#) explains the message security levels used in DWS. If logging output is sent to **syslog** or **eventlog**, severity is translated in DWS as shown in the table.

Table 19-1 Message Severity Levels

| Severity | Description | syslog | eventlog |
|------------|--|---------|-------------|
| debug[1-5] | Provides detailed debug information. | DEBUG | INFORMATION |
| log | Reports information of interest to administrators, for example, checkpoint activity. | INFO | INFORMATION |
| info | Provides information implicitly requested by the user, for example, output from VACUUM VERBOSE . | INFO | INFORMATION |
| notice | Provides information that might be helpful to users, for example, notice of truncation of long identifiers and index created as part of the primary key. | NOTICE | INFORMATION |
| warning | Provides warnings of likely problems, for example, COMMIT outside a transaction block. | NOTICE | WARNING |
| error | Reports an error that causes a command to terminate. | WARNING | ERROR |

| Severity | Description | syslog | eventlog |
|----------|--|--------|----------|
| fatal | Reports the reason that causes a session to terminate. | ERR | ERROR |
| panic | Reports an error that caused all database sessions to terminate. | CRIT | ERROR |

plog_merge_age

Parameter description: Specifies the output interval of performance log data.

Type: SUSET

NOTICE

This parameter value is in milliseconds. You are advised to set this parameter to a value that is a multiple of 1000. That is, the value is in seconds. Name extension of the performance log files controlled by this parameter is .prf. These log files are stored in the **\$GAUSSLOG/gs_profile/<node_name>** directory. *node_name* is the value of **pgxc_node_name** in the **postgres.conf** file. You are advised not to use this parameter externally.

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

- **0** indicates that the current session will not output performance log data.
- A value other than 0 indicates the output interval of performance log data. As the value decreases, more log data is generated, which negatively impacts performance.

Default value: 3s

profile_logging_module

Parameter description: Specifies the type of performance logs. When using this parameter, ensure that the value of **plog_merge_age** is not 0. This parameter is a session-level parameter, and you are not advised to use the **gs_guc** tool to set it. Only clusters of 8.1.3 and later versions support this function.

Type: USERSET

Value range: a string

Default value: OBS, HADOOP and REMOTE_DATANODE are enabled. MD is disabled. You can run the **SHOW profile_logging_module** command to view the value.

Setting method: First, you can run **SHOW profile_logging_module** to view which module is controllable. For example, the query output result is as follows:

```
show profile_logging_module;  
profile_logging_module
```

```
-----  
ALL,on(OBS,HADOOP,REMOTE_DATANODE),off(MD)(1 row)
```

Open the MD performance log and view the setting. The ALL identifier is equivalent to a shortcut operation. That is, logs of all modules can be enabled or disabled.

```
set profile_logging_module='on(md)';  
SET  
  
show profile_logging_module;  
profile_logging_module  
-----  
ALL,on(MD,OBS,HADOOP,REMOTE_DATANODE),off()(1 row)
```

19.10.3 Logging Content

debug_pretty_print

Parameter description: Specifies the logs produced by **debug_print_parse**, **debug_print_rewritten**, and **debug_print_plan**. The output format is more readable but much longer than the output generated when this parameter is set to **off**.

Type: USERSET

Value range: Boolean

- **on** indicates the indentation is enabled.
- **off** indicates the indentation is disabled.

Default value: on

log_duration

Parameter description: Specifies whether to record the duration of every completed SQL statement. For clients using extended query protocols, the time required for parsing, binding, and executing steps are logged independently.

Type: SUSET

Value range: Boolean

- If this parameter is set to **off**, the difference between setting this parameter and setting **log_min_duration_statement** is that when **log_min_duration_statement** is exceeded, the query text is logged, but this parameter does not log it.
- If this parameter is set to **on** and **log_min_duration_statement** has a positive value, all durations are logged but the query text is included only for statements exceeding the threshold. This behavior can be used for gathering statistics in high-load situation.

Default value: on

log_error_verbosity

Parameter description: Specifies the amount of detail written in the server log for each message that is logged.

Type: SUSET

Value range: enumerated values

- **terse** indicates that the output excludes the logging of DETAIL, HINT, QUERY, and CONTEXT error information.
- **verbose** indicates that the output includes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.
- **default** indicates that the output includes the logging of DETAIL, HINT, QUERY, and CONTEXT error information, and excludes the SQLSTATE error code, the source code file name, function name, and number of the line in which the error occurs.

Default value: default

log_lock_waits

Parameter description: If the time that a session used to wait a lock is longer than the value of [deadlock_timeout](#), this parameter specifies whether to record this message in the database. This is useful in determining if lock waits are causing poor performance.

Type: SUSET

Value range: Boolean

- **on** indicates the information is recorded.
- **off** indicates the information is not recorded.

Default value: off

log_statement

Parameter description: Specifies whether to record SQL statements. For clients using extended query protocols, logging occurs when an execute message is received, and values of the Bind parameters are included (with any embedded single quotation marks doubled).

Type: SUSET

NOTICE

Statements that contain simple syntax errors are not logged even if **log_statement** is set to **all**, because the log message is emitted only after basic parsing has been completed to determine the statement type. If the extended query protocol is used, this setting also does not log statements before the execution phase (during parse analysis or planning). Set **log_min_error_statement** to ERROR or lower to log such statements.

Value range: enumerated values

- **none** indicates that no statement is recorded.

- **ddl** indicates that all data definition statements, such as CREATE, ALTER, and DROP, are recorded.
- **mod** indicates that all DDL statements and data modification statements, such as INSERT, UPDATE, DELETE, TRUNCATE, and COPY FROM, are recorded.
- **all** indicates that all statements are recorded. The PREPARE, EXECUTE, and EXPLAIN ANALYZE statements are also recorded.

Default value: none

log_temp_files

Parameter description: Specifies whether to record the delete information of temporary files. Temporary files can be created for sorting, hashing, and temporary querying results. A log entry is generated for each temporary file when it is deleted.

Type: SUSET

Value range: an integer ranging from -1 to INT_MAX. The unit is KB.

- A positive value indicates that the delete information of temporary files whose values are larger than that of **log_temp_files** is recorded.
- If the parameter is set to **0**, all the delete information of temporary files is recorded.
- If the parameter is set to **-1**, the delete information of no temporary files is recorded.

Default value: -1

logging_module

Parameter description: Specifies whether module logs can be output on the server. This parameter is a session-level parameter, and you are not advised to use the **gs_guc** tool to set it.

Type: USERSET

Value range: a string

Default value: off. All the module logs on the server can be viewed by running **show logging_module**.

Setting method: First, you can run **show logging_module** to view which module is controllable. For example, the query output result is as follows:

```
show logging_module;
logging_module
-----
ALL,on(),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,SP
ACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,
OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCPACCELERATE,PLANHINT,PARQUET,CARB
ONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,CO
MM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTSCAN)
(1 row)
```

Controllable modules are identified by uppercase letters, and the special ID ALL is used for setting all module logs. You can control module logs to be exported by setting the log modules to **on** or **off**. Enable log output for SSL:

```
set logging_module='on(SSL)';
SET
show
logging_module;
```

| logging_module |
|--|
| ALL,on(SSL),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,GDS,TBLSPC,WLM,SPACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCAN) |

(1 row)

SSL log output is enabled.

The ALL identifier is equivalent to a shortcut operation. That is, logs of all modules can be enabled or disabled.

```
set logging_module='off(ALL)';
SET
show
logging_module;
```

| logging_module |
|---|
| ALL,on(),off(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,SPACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCAN) |

(1 row)

```
set logging_module='on(ALL)';
SET
show
logging_module;
```

| logging_module |
|---|
| ALL,on(DFS,GUC,HDFS,ORC,SLRU,MEM_CTL,AUTOVAC,ANALYZE,CACHE,ADIO,SSL,GDS,TBLSPC,WLM,SPACE,OBS,EXECUTOR,VEC_EXECUTOR,STREAM,LLVM,OPT,OPT_REWRITE,OPT_JOIN,OPT_AGG,OPT_SUBPLAN,OPT_SETOP,OPT_CARD,OPT_SKEW,SMP,UDF,COOP_ANALYZE,WLMCP,ACCELERATE,PLANHINT,PARQUET,CARBONDATA,SNAPSHOT,XACT,HANDLE,CLOG,TQUAL,EC,REMOTE,CN_RETRY,PLSQL,TEXTSEARCH,SEQ,INSTR,COMM_IPC,COMM_PARAM,CSTORE,JOB,STREAMPOOL,STREAM_CTESCAN),off() |

(1 row)

COMM_IPC logs must be enabled or disabled explicitly. You can run either of the following command to enable the log function of COMM_IPC:

```
set logging_module='on(ALL)';  
SET  
set logging_module='on(COMM_IPC)';  
SET
```

After the setting is performed, the log function of the COMM_IPC module will not be automatically disabled. To disable the log function of the COMM_IPC module, you must run the following commands:

```
set logging_module='off(ALL)';  
SET  
set logging_module='off(COMM_IPC)';  
SET
```

Dependency relationship: The value of this parameter depends on the settings of [log_min_messages](#).

enable_unshipping_log

Parameter description: Specifies whether to log statements that are not pushed down. The logs help locate performance issues that may be caused by statements not pushed down.

Type: SUSET

Value range: Boolean

- **on:** Statements not pushed down will be logged.
- **off:** Statements not pushed down will not be logged.

Default value: on

log_statement_filter_list

Parameter description: Specifies whether to record SQL statements. It sets a collection of error codes, with multiple error codes separated by commas, for example: **GS_001, GS_002**. No SQL statement is recorded in the error code log. This parameter is supported only by 9.1.0.200 and later versions.

Type: SUSET

Value range: a string

Default value: an empty string

19.11 Runtime Statistics

19.11.1 Query and Index Statistics Collector

The query and index statistics collector is used to collect statistics during database running. The statistics include the times of inserting and updating a table and an index, the number of disk blocks and tuples, and the time required for the last cleanup and analysis on each table. The statistics can be viewed by querying system view families `pg_stats` and `pg_statistic`. The following parameters are used to set the statistics collection feature in the server scope.

track_activities

Parameter description: Collects statistics about the commands that are being executed in session.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: on

track_counts

Parameter description: Collects statistics about data activities.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

NOTE

When the database to be cleaned up is selected from the AutoVacuum automatic cleanup process, the database statistics are required. In this case, the default value is set to **on**.

Default value: on

track_io_timing

Parameter description: Collects statistics about I/O invoking timing in the database. The I/O timing statistics can be queried by using the **pg_stat_database** parameter.

Type: SUSET

Value range: Boolean

- If this parameter is set to **on**, the collection function is enabled. In this case, the collector repeatedly queries the OS at the current time. As a result, large numbers of costs may occur on some platforms. Therefore, the default value is set to **off**.
- **off** indicates that the statistics collection function is disabled.

Default value: off

track_functions

Parameter description: Collects statistics about invoking times and duration in a function.

Type: SUSET

NOTICE

When the SQL functions are set to inline functions queried by the invoking, these SQL functions cannot be traced no matter these functions are set or not.

Value range: enumerated values

- **pl** indicates that only procedural language functions are traced.
- **all** indicates that SQL and C language functions are traced.
- **none** indicates that the function tracing function is disabled.

Default value: none

update_process_title

Parameter description: Collects statistics updated with a process name each time the server receives a new SQL statement.

The process name can be viewed on Windows task manager by running the **ps** command.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: off

track_thread_wait_status_interval

Parameter description: Specifies the interval of collecting the thread status information periodically.

Type: SUSET

Value range: an integer ranging from 0 to 1440, in minutes.

Default value: 30min

enable_save_datachanged_timestamp

Parameter description: Specifies whether to record the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE/TRUNCATE/DROP PARTITION** is performed on table data.

Type: USERSET

Value range: Boolean

- **on** indicates that the time when an operation is performed on table data will be recorded.
- **off** indicates that the time when an operation is performed on table data will not be recorded.

Default value: on

enable_save_dataaccess_timestamp

Parameter description: Specifies whether to record the last access time of a table. This parameter is supported only by 8.2.1.210 and later cluster versions.

Type: USERSET

Value range: Boolean

- **on** indicates that the last access time of the table is recorded.
- **off** indicates that the last access time of the table is not recorded.

Default value: off

instr_unique_sql_count

Parameter description: Specifies whether to collect Unique SQL statements and the maximum number allowed.

Type: SIGHUP

Value range: an integer ranging from 0 to INT_MAX

- If it is set to **0**, Unique SQL statistics are not collected.
- If the value is greater than **0**, the number of Unique SQL statements collected on the CN cannot exceed the value of this parameter. When the number of collected Unique SQL statements reaches the upper limit, the collection is stopped. In this case, you can increase the value of **reload** to continue the collection.

Default value: 0



CAUTION

If a new value is smaller than the original value, the Unique SQL statistics collected on the CN will be cleared.

track_sql_count

Parameter description: Specifies whether to collect statistics on the number of the **SELECT**, **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** statements that are being executed in each session, the response time of the **SELECT**, **INSERT**, **UPDATE**, and **DELETE** statements, and the number of DDL, DML, and DCL statements.

Type: SUSET

Value range: Boolean

- **on** indicates that the statistics collection function is enabled.
- **off** indicates that the statistics collection function is disabled.

Default value: on

 NOTE

- The **track_sql_count** parameter is restricted by the **track_activities** parameter.
 - If **track_activities** is set to **on** and **track_sql_count** is set to **off**, a warning message indicating that **track_sql_count** is disabled will be displayed when the view **gs_sql_count**, **pgxc_sql_count**, **gs_workload_sql_count**, **pgxc_workload_sql_count**, **global_workload_sql_count**, **gs_workload_sql_elapse_time**, **pgxc_workload_sql_elapse_time**, or **global_workload_sql_elapse_time** are queried.
 - If both **track_activities** and **track_sql_count** are set to **off**, two logs indicating that **track_activities** is disabled and **track_sql_count** is disabled will be displayed when the views are queried.
 - If **track_activities** is set to **off** and **track_sql_count** is set to **on**, a log indicating that **track_activities** is disabled will be displayed when the views are queried.
- If this parameter is disabled, querying the view returns **0**.

enable_parallel_analyze

Parameter description: Specifies whether to use parallel sampling for internal and foreign table analysis. This parameter is supported only by clusters of version 9.1.0 or later.

Type: USERSET

Value range: Boolean

- **true** indicates that parallel sampling is used for internal and foreign table analysis.
- **false** indicates that parallel sampling is not used for internal and foreign table analysis.

Default value: true

 CAUTION

- When **enable_parallel_analyze** is set to **true** and analyzing foreign tables, try to avoid adding NOT NULL constraints to the target foreign table columns to prevent constraint failure due to data source changes. Currently, parallel sampling does not support materialized views. If analyze fails due to such reasons, set this parameter to **false**.
 - Currently, parallel sampling only supports analyzing ordinary column-store internal tables. This optimization does not take effect when the internal table uses hstore/hstore_opt or is declared as a replicated table.
 - Currently, parallel sampling only supports analyzing foreign tables stored in parquet/orc format. This optimization does not take effect when the foreign table is in another format.
-

parallel_analyze_workers

Parameter description: Specifies the number of concurrent threads for parallel analyze sampling. This parameter is supported only by clusters of version 9.1.0 or later.

Type: USERSET

Value range: an integer ranging from 0 to 64

Default value: 10

 NOTE

The value of this parameter should correspond to the cluster load. When the cluster load is low, you can increase the parameter value appropriately based on the cluster configuration to further improve the efficiency of analyze execution.

analyze_sample_multiplier

Parameter description: Specifies the multiplier for the stripe sampling rate used in analyzing foreign tables. This parameter is supported only by clusters of version 9.1.0 or later.

Type: SUSERSET

Value range: an integer ranging from 0 to 100. 0 indicates that the stripe sampling rate is 100%.

Default value: 3

19.11.2 Performance Statistics

During database operation, accessing locks, disk I/O operations, and handling invalid messages can all be performance bottlenecks for the database. DWS provides performance statistics methods that can help easily locate performance issues.

Generating Performance Statistics Logs

Parameter description: For each query, the following four parameters control the performance statistics of corresponding modules recorded in the server log:

- The **og_parser_stats** parameter controls the performance statistics of a parser recorded in the server log.
- The **log_planner_stats** parameter controls the performance statistics of a query optimizer recorded in the server log.
- The **log_executor_stats** parameter controls the performance statistics of an executor recorded in the server log.
- The **log_statement_stats** parameter controls the performance statistics of the whole statement recorded in the server log.

All these parameters can only provide assistant analysis for administrators, which are similar to the `getrusage()` of the Linux OS.

Type: SUSERSET

NOTICE

- **log_statement_stats** records the total statement statistics while other parameters only record statistics about each statement.
- The **log_statement_stats** parameter cannot be enabled together with other parameters recording statistics about each statement.

Value range: Boolean

- **on** indicates the function of recording performance statistics is enabled.
- **off** indicates the function of recording performance statistics is disabled.

Default value: off

19.12 Resource Management

If database resource usage is not controlled, concurrent tasks easily preempt resources. As a result, the OS will be overloaded and cannot respond to user tasks; or even crash and cannot provide any services to users. The DWS workload management function balances the database workload based on available resources to avoid database overloading.

space_once_adjust_num

Parameter description: In the space control and space statistics functions, specifies the threshold of the number of files processed each time during slow building and fine-grained calibration. This parameter is supported only by clusters of version 8.1.3 or later.

Type: SIGHUP

Value range: an integer ranging from 0 to INT_MAX

- The value **0** indicates that the slow build and fine-grained calibration functions are disabled.

Default value: 300

 NOTE

The file quantity threshold affects database resources. You are advised to set the threshold to a proper value.

default_partition_cache_strategy

Parameter description: Specifies the default policy for controlling partition caching. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: enumerated values

- **cache_each_partition_as_possible** enables maximum data caching. Data may not be written to CUs when being inserted into different partitions.

- **flush_when_switch_partition** indicates that data is written to CUs if the data belongs to different partitions during insertion.

Default value: `cache_each_partition_as_possible`

max_active_statements

Parameter description: Specifies the maximum global concurrency. This parameter applies to a job on a CN.

The database administrator changes the value of this parameter based on system resources (for example, CPU, I/O, and memory resources) so that the system fully supports the concurrency tasks and avoids too many concurrency tasks resulting in system crash.

Type: SIGHUP

Value range: an integer ranging from -1 to INT_MAX. The values -1 and 0 indicate that the number of concurrent requests is not limited.

Default value: 60

cgroup_name

Parameter description: Specifies the name of the Cgroup in use. It can be used to change the priorities of jobs in the queue of a Cgroup.

If you set **cgroup_name** and then **session_respool**, the Cgroups associated with **session_respool** take effect. If you reverse the order, Cgroups associated with **cgroup_name** take effect.

If the Workload Cgroup level is specified during the **cgroup_name** change, the database does not check the Cgroup level. The level ranges from 1 to 10.

Type: USERSET

You are not advised to set **cgroup_name** and **session_respool** at the same time.

Value range: a string

Default value: `DefaultClass:Medium`

NOTE

DefaultClass:Medium indicates the **Medium** Cgroup belonging to the **Timeshare** Cgroup under the **DefaultClass** Cgroup.

enable_cgroup_switch

Parameter description: Specifies whether the database automatically switches to the **TopWD** group when executing statements by group type.

Type: USERSET

Value range: Boolean

- **on:** The database automatically switches to the **TopWD** group when executing statements by group type.

- **off**: The database does not automatically switch to the **TopWD** group when executing statements by group type.

Default value: off

memory_tracking_mode

Parameter description: Specifies the memory information recording mode.

Type: USERSET

Value range:

- **none**: Memory statistics is not collected.
- **normal**: Only memory statistics is collected in real time and no file is generated.
- **executor**: The statistics file is generated, containing the context information about all allocated memory used by the execution layer.
- **fullexec**: The generated file includes the information about all memory contexts requested by the execution layer.

Default value: none

memory_detail_tracking

Parameter description: Specifies the sequence number of the memory background information distributed in the needed thread and **plannodeid** of the query where the current thread is located.

Type: USERSET

Value range: a string

Default value: empty

NOTICE

It is recommended that you retain the default value for this parameter.

enable_resource_track

Parameter description: Specifies whether the real-time resource monitoring function is enabled. This parameter must be applied on both CNs and DN.

Type: SIGHUP

Value range: Boolean

- **on** indicates the resource monitoring function is enabled.
- **off** indicates the resource monitoring function is disabled.

Default value: on

enable_resource_record

Parameter description: Specifies whether resource monitoring records are archived. When this parameter is enabled, records that have been executed are archived to the corresponding **INFO** views (**GS_WLM_SESSION_INFO** and **GS_WLM_OPERATOR_INFO**). This parameter must be applied on both CNs and DNs.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the resource monitoring records are archived.
- **off** indicates that the resource monitoring records are not archived.

Default value: on

NOTE

The default value of this parameter is **on** for a new cluster. In upgrade scenarios, the default value of this parameter is the same as that of the source version.

enable_track_record_subsql

Parameter description: Specifies whether to enable the function of recording and archiving sub-statements. When this function is enabled, sub-statements in stored procedures and anonymous blocks are recorded and archived to the corresponding **INFO** table (**GS_WLM_SESSION_INFO**). This parameter is a session-level parameter. It can be configured and take effect in the session connected to the CN and affects only the statements in the session. It can also be configured on both the CN and DN and take effect globally.

Type: USERSET

Value range: Boolean

- **on** indicates that the sub-statement resource monitoring records are archived.
- **off** indicates that the sub-statement resource monitoring records are not archived.

Default value: on

block_rule_cost

Parameter description: Specifies the minimum cost to trigger a query filter. This is supported only by clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX

- Value **-1** indicates that the system filters all statements without considering the cost.
- Value **0** indicates that all statements whose cost is greater than 0 are intercepted, but special statements (whose cost is 0) are not intercepted.
- If the value is greater than **0** and the statement cost is less than the value of **block_rule_cost**, the statement is not intercepted; otherwise, it is intercepted.

Default value: -1

enable_user_metric_persistent

Parameter description: Specifies whether the user historical resource monitoring dumping function is enabled. When this function is enabled, data in the [PG_TOTAL_USER_RESOURCE_INFO](#) view is periodically sampled and saved to the [GS_WLM_USER_RESOURCE_HISTORY](#) system catalog, and data in the [GS_RESPOOL_RESOURCE_INFO](#) view is periodically sampled and saved to the [GS_RESPOOL_RESOURCE_HISTORY](#) system catalog.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the user historical resource monitoring dumping function is enabled.
- **off** indicates that the user historical resource monitoring dumping function is disabled.

Default value: on

user_metric_retention_time

Parameter description: Specifies the retention time of the user historical resource monitoring data.

Type: SIGHUP

Value range: an integer ranging from 0 to 3650. The unit is day.

- If this parameter is set to **0**, user historical resource monitoring data is permanently stored.
- If the value is greater than **0**, user historical resource monitoring data is stored for the specified number of days.

Default value: 7

resource_track_level

Parameter description: Specifies the resource monitoring level of the current session. This parameter is valid only when **enable_resource_track** is set to **on**.

Type: USERSET

Value range: enumerated values

- **none:** Resources are not monitored.
- **query:** enables query-level resource monitoring. If this function is enabled, the plan information (similar to the output information of EXPLAIN) of SQL statements will be recorded in top SQL statements.
- **perf:** enables the perf-level resource monitoring. If this function is enabled, the plan information (similar to the output information of EXPLAIN ANALYZE) that contains the actual execution time and the number of execution rows will be recorded in the top SQL.

- **operator_realtime:** enables the operator-level resource monitoring. If this function is enabled, the operator information of jobs running in real time is recorded in the top SQL statements, but is not persisted to the historical top SQL statements.
- **operator:** enables the operator-level resource monitoring. If this function is enabled, not only the information including the actual execution time and number of execution rows is recorded in the top SQL statement, but also the operator-level execution information is updated to the top SQL statement.

Default value: query

fast_obs_tablesize_method

Parameter description: Specifies the method for quickly calculating the size of column-store V3 and V3 HStore Opt tables. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: enumerated values

- **0:** The table size is calculated by listing OBS files.
- **1:** The table size is calculated through WLM background statistics using **pg_relfilenode_size**.
- **2:** The table size is estimated by calculating the maximum offset of each file in cudesc.

Default value: 2

fast_obs_dbsize_method

Parameter description: Specifies the method for quickly calculating the size of database data on OBS. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: enumerated values

- **0:** The size of the database is directly estimated based on the OBS bucket.
- **1:** The size of the entire database is normally calculated in regular mode.

Default value: 0

resource_track_cost

Parameter description: Specifies the minimum execution cost for resource monitoring on statements in the current session. This parameter is valid only when **enable_resource_track** is set to **on**.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX

- **-1** indicates that resource monitoring is disabled.

- A value greater than or equal to **0** indicates that statements whose execution cost exceeds this value will be monitored.

Default value: 0

 NOTE

The default value of this parameter is **0** for a new cluster. In upgrade scenarios, the default value of this parameter is the same as that of the source version.

resource_track_duration

Parameter description: Specifies the minimum statement execution time that determines whether information about jobs of a statement recorded in the real-time view (see [Table 15-1](#)) will be dumped to a historical view after the statement is executed. Job information will be dumped from the real-time view (with the suffix **statistics**) to a historical view (with the suffix **history**) if the statement execution time is no less than this value. This parameter is valid only when **enable_resource_track** is set to **on**.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX. The unit is second (s).

- **0** indicates that information about all statements recorded in the real-time resource monitoring view (see [Table 15-1](#)) will be archived into historical views.
- If the value is greater than **0**, the system archives historical information if the total execution and queuing time of statements in the real-time resource monitoring view ([Table 15-1](#)) goes over the parameter value.

Default value: 60s

resource_track_subsql_duration

Parameter description: Filters the minimum execution time of substatements in a stored procedure. This parameter is supported only by clusters of version 8.2.1 or later.

If the execution time of a sub-statement in a stored procedure is greater than the value of this parameter, the job information is dumped to the top SQL archive table. This parameter takes effect only when **enable_track_record_subsql** is set to **on**.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX. The unit is second (s).

- If the value is **0**, historical information about all substatements in the stored procedure is archived.
- If the value is greater than **0**, historical information is archived when the execution time of a substatement in a stored procedure exceeds the value of this parameter.

Default value: 180s

query_exception_count_limit

Parameter description: Specifies the maximum number of times that a job triggers an exception rule. If the number of times that a job triggers an exception rule reaches the upper limit, the job will be automatically added to the blocklist and cannot be executed anymore. The job can be resumed only after it is removed from the blocklist.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX

- If the value is **-1**, the number of times that a job triggers an exception rule is not limited. That is, the job will not be automatically added to blocklist even if it triggers an exception rule for many times.
- If the value is greater than or equal to **0**, the job will be added to the blocklist immediately when the number of times it triggers an exception rule reaches the threshold. The values **0** and **1** both indicate that a job is added to blocklist once the job triggers an exception rule.

Default value: -1

disable_memory_protect

Parameter description: Stops memory protection. To query system views when system memory is insufficient, set this parameter to **on** to stop memory protection. This parameter is used only to diagnose and debug the system when system memory is insufficient. Set it to **off** in other scenarios.

Type: USERSET

Value range: Boolean

- **on** indicates that memory protection stops.
- **off** indicates that memory is protected.

Default value: off

query_band

Parameter description: Specifies the job type of the current session.

Type: USERSET

Value range: a string

Default value: empty

 CAUTION

Pay attention to the following when modifying this parameter:

1. When this parameter is disabled, it means that the user does not need CCN control function, and the CCN memory negative feedback mechanism will be invalid.
2. When a job is running, if the value of GUC is changed from **OFF** to **ON**, the CCN memory negative feedback mechanism takes effect. If the concurrency is high, the memory may be temporarily unavailable. After the running job is done, the dynamic load function recovers.
3. When a job is running and most jobs are delivered by users from the default resource pool, you are not advised to change the GUC parameter from **enabled** to **disabled**. It may cause a memory error. When there is no job delivered by users from the default resource pool, then you can change the parameter. You are advised to bind a user resource pool before delivering jobs.

wlm_memory_feedback_adjust

Parameter description: Specifies whether to enable memory negative feedback in dynamic load management. (This parameter is supported only by clusters of version 8.2.0 or later.)

Memory is preempted based on the estimated statement memory usage calculated on the CN. If the estimated memory usage of a statement is too high, it will preempt too much memory, causing subsequent jobs to be queued. With the negative memory feedback mechanism, if the cluster memory usage has been overestimated for a period of time, the CCN node will dynamically release some memory for subsequent jobs, improving resource utilization.

Type: SIGHUP

Value range: a string

- **on** indicates that memory negative feedback is enabled.
- **off** indicates that memory negative feedback is disabled.
- **on()** enables the memory negative feedback function and specifies the time and estimated memory percentage parameter required to trigger the negative feedback. For example, **on(60,50)** indicates that to trigger the negative feedback mechanism, the memory must be overestimated for 60 consecutive seconds, and the preempted memory needs must exceed 50% of the available memory. By default, the wait time before the negative feedback mechanism takes effect is 50 seconds. The minimum estimated total memory usage for triggering the mechanism is over 40% of the available system memory.

Default value: on

bbox_dump_count

Parameter description: Specifies the maximum number of core files that are generated by DWS and can be stored in the path specified by **bbox_dump_path**. If the number of core files exceeds this value, old core files will be deleted. This parameter is valid only if **enable_bbox_dump** is set to **on**.

Type: USERSET

Value range: an integer ranging from 1 to 20

Default value: 8

 NOTE

When core files are generated during concurrent SQL statement execution, the number of files may be larger than the value of **bbox_dump_count**.

io_limits

Parameter description: This parameter has been discarded in version 8.1.2 and is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

Type: USERSET

Value range: an integer ranging from 0 to 1073741823

Default value: 0

io_priority

Parameter description: This parameter has been discarded in version 8.1.2 and is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

Type: USERSET

Value range: enumerated values

- None
- Low
- Medium
- High

Default value: None

session_respool

Parameter description: Specifies the resource pool associated with the current session.

Type: USERSET

If you set **cgroup_name** and then **session_respool**, the Cgroups associated with **session_respool** take effect. If you reverse the order, Cgroups associated with **cgroup_name** take effect.

If the Workload Cgroup level is specified during the **cgroup_name** change, the database does not check the Cgroup level. The level ranges from 1 to 10.

You are not advised to set **cgroup_name** and **session_respool** at the same time.

Value range: a string. This parameter can be set to the resource pool configured through **create resource pool**.

Default value: `invalid_pool`

enable_transaction_parctl

Parameter description: whether to control transaction block statements and stored procedure statements.

Type: USERSET

Value range: Boolean

- **on:** Transaction block statements and stored procedure statements are controlled.
- **off:** Transaction block statements and stored procedure statements are not controlled.

Default value: `on`

session_history_memory

Parameter description: Specifies the memory size of a historical query view.

Type: SIGHUP

Value range: an integer ranging from 10 MB to 50% of `max_process_memory`

Default value: `100MB`

topsql_retention_time

Parameter description: Specifies the retention period of historical Top SQL data in the `gs_wlm_session_info` and `gs_wlm_operator_info` tables.

Type: SIGHUP

Value range: an integer ranging from 0 to 3650. The unit is day.

- If it is set to `0`, the data is stored permanently.
- If the value is greater than `0`, the data is stored for the specified number of days.

Default value: `30`



CAUTION

- Before setting this GUC parameter to enable the data retention function, delete data from the `gs_wlm_session_info` and `gs_wlm_operator_info` tables.
 - The default value of this parameter is `30` for a new cluster. In upgrade scenarios, the default value of this parameter is the same as that of the source version.
-

transaction_pending_time

Parameter description: maximum queuing time of transaction block statements and stored procedure statements if `enable_transaction_parctl` is set to `on`.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX. The unit is second (s).

- **-1 or 0:** No queuing timeout is specified for transaction block statements and stored procedure statements. The statements can be executed when resources are available.
- **Value greater than 0:** If transaction block statements and stored procedure statements have been queued for a time longer than the specified value, they are forcibly executed regardless of the current resource situation.

Default value: 0

NOTICE

This parameter is valid only for internal statements of stored procedures and transaction blocks. That is, this parameter takes effect only for the statements whose **enqueue** value (for details, see [PG_SESSION_WLMSTAT](#)) is **Transaction** or **StoredProc**.

enable_concurrency_scaling

Parameter description: Specifies whether to enable the elastic concurrent expansion function. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the elastic concurrent expansion function is enabled.
- **off** indicates that the elastic concurrent expansion function is disabled.

Default value: off

concurrency_scaling_max_idle_time

Parameter description: Specifies the maximum idle time of the elastic logical cluster created for concurrent expansion. If it exceeds the set value of this parameter, it will enter the process of destroying the elastic logical cluster. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: SIGHUP

Value range: an integer ranging from 0 to 60, in minutes. The value **0** indicates that the elastic logical cluster created during concurrent expansion will not be destroyed. Exercise caution when setting this value.

Default value: 5

concurrency_scaling_limit_per_main_vw

Parameter description: Specifies the maximum number of concurrent elastic logical clusters that can be created for each classic logical cluster. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: SIGHUP

Value range: an integer ranging from 0 to 32

Default value: 5

concurrency_scaling_max_vw_active_statements

Parameter description: Specifies the maximum number of concurrent tasks that can be executed in an elastic logical cluster. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: SIGHUP

Value range: an integer ranging from -1 to INT_MAX

- Value **0** indicates that the elastic logical cluster does not execute any jobs. Exercise caution when setting this value.
- **-1** indicates that the elastic logical cluster is not under concurrency control. Exercise caution when setting this value. This value is supported only by clusters of version 9.1.0.200 or later.

Default value: 60

concurrency_scaling_max_waiting_statements

Parameter description: Specifies the number of queued elastic jobs in the global queue that trigger the creation process of the elastic logical cluster for concurrent expansion. This parameter is supported only by clusters of version 9.1.0.100 or later.

If the number of queued elastic jobs is greater than or equal to the value set for this parameter, and the number of concurrently created elastic logical clusters does not exceed the value set for **concurrency_scaling_limit_per_main_vw**, the process of creating a concurrent expansion elastic logical cluster will be triggered automatically.

Type: SIGHUP

Value range: an integer ranging from 0 to INT_MAX. The value **0** indicates that an elastic logical cluster is created even if there are no queued jobs. If set to **0**, a large number of resources are consumed. Exercise caution when setting this value.

Default value: 10

19.13 Automatic Cleanup

The automatic cleanup process in the system automatically runs the **VACUUM** and **ANALYZE** statements to reclaim the record space marked as deleted and update statistics in the table.

autovacuum_compaction_rows_limit

Parameter description: Specifies the small CU threshold. A CU whose number of live tuples is less than the value of this parameter is considered as a small CU.

Type: USERSET

Value range: an integer ranging from -1 to 5000

Default value: 2500



CAUTION

- **You are advised not to modify this parameter. If you need to modify this parameter, contact technical support.**
 - If the version is earlier than 9.1.0.100, do not set this parameter, or there may be duplicate data in primary keys.
 - If the version is earlier than 9.1.0.100, value **-1** indicates that the 0 CU switch is disabled.
 - In version 9.1.0.100, the default value of this parameter is **0**.
 - In 9.1.0.200 and later versions, the default value of this parameter is **2500**.
-

autoanalyze_mode

Parameter description: Specifies the autoanalyze mode. This parameter is supported by clusters of version 8.2.0 or later.

Type: USERSET

Value range: enumerated values

- **normal** indicates common autoanalyze.
- **light** indicates lightweight autoanalyze.

Default value:

- If the current cluster is upgraded from an earlier version to 8.2.0, the default value is **normal** to ensure forward compatibility.
- If the cluster version 8.2.0 is newly installed, the default value is **light**.

analyze_stats_mode

Parameter description: Specifies the mode for **ANALYZE** to calculate statistics.

Type: USERSET

Value range: enumerated values

- **memory** indicates that the memory is forcibly used to calculate statistics. Multi-column statistics are not calculated.
- **sample_table** indicates that temporary sampling tables are forcibly used to calculate statistics. Temporary tables do not support this mode.
- **dynamic** indicates that the statistics calculation mode is determined based on the size of **maintenance_work_mem**. If **maintenance_work_mem** can store samples, the memory mode is used. Otherwise, the temporary sampling table mode is used.

Default value:

- If the current cluster is upgraded from an earlier version to 8.2.0.100, the default value is **memory** to ensure forward compatibility.
- If the cluster version 8.2.0.100 is newly installed, the default value is **dynamic**.

analyze_sample_mode

Parameter description: Specifies the sampling model used by **ANALYZE**.

Type: USERSET

Value range: an integer ranging from **0** to **2**

- **0** indicates the default reservoir sampling.
- **1** indicates the optimized reservoir sampling.
- **2** indicates range sampling.

Default value: 0

autovacuum_max_workers

Parameter description: Specifies the maximum number of automatic cleanup threads running at the same time.

Type: SIGHUP

Value range: an integer ranging from 0 to 128. **0** indicates that **autovacuum** is disabled.

Default value: 6

NOTE

This parameter works with autovacuum. The rules for clearing system catalogs and user tables are as follows:

- When **autovacuum_max_workers** is set to **0**, **autovacuum** is disabled and no tables are cleared.
- When **autovacuum_max_workers** is set to a value greater than 0 and **autovacuum** is set to **off**, the system only clears the system catalogs and column-store tables with delta tables enabled (such as vacuum delta tables, vacuum cudesc tables, and delta merge).
- When **autovacuum_max_workers** is set to a value greater than 0 and **autovacuum** is set to **on**, all tables will be cleared.

autovacuum_max_workers_hstore

Parameter description: Specifies the maximum number of concurrent automatic cleanup threads used for hstore tables in **autovacuum_max_workers**.

Type: SIGHUP

Value range: an integer ranging from 0 to 128. **0** indicates that the automatic cleanup function of HStore tables is disabled.

Default value: 3

 NOTE

To use HStore tables, set the following parameters, or the HStore performance will deteriorate severely. The recommended settings are as follows:

autovacuum_max_workers_hstore=3, autovacuum_max_workers=6, autovacuum=true

autovacuum_naptime

Parameter description: Specifies the interval between two automatic cleanup operations.

Type: SIGHUP

Value range: an integer ranging from 1 to 2147483. The unit is second.

Default value: 60s

autovacuum_vacuum_cost_delay

Parameter description: Specifies the value of the cost delay used in the **autovacuum** operation.

Type: SIGHUP

Value range: an integer ranging from -1 to 100. The unit is ms. -1 indicates that the normal vacuum cost delay is used.

Default value: 2ms

check_crossvw_write

Parameter description: Specifies whether to enable cross-VW write detection. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: USERSET

Value range: an integer, -1 or 1.

- The value -1 indicates that it is compatible with the capabilities of version 9.0.3. For the v3 table vacuum, it only clears non-last files for all epochs.
- The value 1 indicates checking whether it is a cross-VW write scenario. For the v3 table vacuum, if it is determined to be a non-cross-VW write scenario, it clears non-last files for all epochs, clears the last file for the current epoch, and clears the last file for epochs that are less than the current epoch. If it is determined to be a cross-VW write scenario, CNs will obtain epoch information from all DN and package it into an epochList to be sent to the metadata VW. The v3 table vacuum will clear non-last files for all epochs and clear the last file for epochs that are less than max{epochList} and not in epochList.

Default value: 1

enable_pg_stat_object

Parameter description: Specifies whether **AUTO VACUUM** updates the **PG_STAT_OBJECT** system catalog. This parameter is supported only by clusters of version 8.2.1 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the **PG_STAT_OBJECT** system catalog is updated during **AUTO VACUUM**.
- **off** indicates that the **PG_STAT_OBJECT** system catalog is not updated during **AUTO VACUUM**.

Default value: on

19.14 Default Settings of Client Connection

19.14.1 Statement Behavior

This section describes related default parameters involved in the execution of SQL statements.

search_path

Parameter description: Specifies the order in which schemas are searched when an object is referenced with no schema specified. The value of this parameter consists of one or more schema names. Different schema names are separated by commas (,).

Type: USERSET

- If the schema of a temporary table exists in the current session, the scheme can be listed in **search_path** by using the alias **pg_temp**, for example, '**pg_temp,public**'. The schema of a temporary table has the highest search priority and is always searched before all the schemas specified in **pg_catalog** and **search_path**. Therefore, do not explicitly specify **pg_temp** to be searched after other schemas in **search_path**. This setting will not take effect and an error message will be displayed. If the alias **pg_temp** is used, the temporary schema will be only searched for database objects, including tables, views, and data types. Functions or operator names will not be searched for.
- The schema of a system catalog, **pg_catalog**, has the second highest search priority and is the first to be searched among all the schemas, excluding **pg_temp**, specified in **search_path**. Therefore, do not explicitly specify **pg_catalog** to be searched after other schemas in **search_path**. This setting will not take effect and an error message will be displayed.
- When an object is created without specifying a particular schema, the object will be placed in the first valid schema listed in **search_path**. An error will be reported if the search path is empty.
- The current effective value of the search path can be examined through the SQL function **current_schema**. This is different from examining the value of **search_path**, because the **current_schema** function displays the first valid schema name in **search_path**.

Value range: a string

 NOTE

- When this parameter is set to "**\$user**", **public**, a database can be shared (where no users have private schemas, and all share use of public), and private per-user schemas and combinations of them are supported. Other effects can be obtained by modifying the default search path setting, either globally or per-user.
- When this parameter is set to a null string (""), the system automatically converts it into a pair of double quotation marks ("").
- If the content contains double quotation marks, the system considers them as insecure characters and converts each double quotation mark into a pair of double quotation marks.

Default value: "**\$user**",**public**

 NOTE

\$user indicates the name of the schema with the same name as the current session user. If the schema does not exist, **\$user** will be ignored.

current_schema

Parameter description: Specifies the current schema.

Type: USERSET

Value range: a string

Default value: "**\$user**",**public**

 NOTE

\$user indicates the name of the schema with the same name as the current session user. If the schema does not exist, **\$user** will be ignored.

default_tablespace

Parameter description: Specifies the default tablespace of the created objects (tables and indexes) when a **CREATE** command does not explicitly specify a tablespace.

- The value of this parameter is either the name of a tablespace, or an empty string that specifies the use of the default tablespace of the current database. If a non-default tablespace is specified, users must have CREATE privilege for it. Otherwise, creation attempts will fail.
- This parameter is not used for temporary tables. For them, the [temp_tablespaces](#) is consulted instead.
- This parameter is not used when users create databases. By default, a new database inherits its tablespace setting from the template database.

Type: USERSET

Value range: a string. An empty string indicates that the default tablespace is used.

Default value: empty

default_storage_nodegroup

Parameter description: Specifies the Node Group where a table is created by default. This parameter takes effect only for ordinary tables.

Type: USERSET

Value range: a string

- **installation:** indicates that the table is created in the installed Node Group by default.
- **random_node_group:** indicates that the table is created in a randomly selected Node Group by default. This feature is supported in 8.1.2 or later and is used only in the test environment.
- **roach_group:** indicates that the table is created in all nodes by default. This value is reserved for the Roach tool and cannot be used in other scenarios.
- A value other than the preceding three options indicates that the table is created in a specified Node Group.

Default value: installation

temp_tablespaces

Parameter description: Specifies tablespaces to which temporary objects will be created (temporary tables and their indexes) when a **CREATE** command does not explicitly specify a tablespace. Temporary files for sorting large data are created in these tablespaces.

The value of this parameter is a list of names of tablespaces. When there is more than one name in the list, DWS chooses a random tablespace from the list upon the creation of a temporary object each time. Except that within a transaction, successively created temporary objects are placed in successive tablespaces in the list. If the element selected from the list is an empty string, DWS will automatically use the default tablespace of the current database instead.

Type: USERSET

Value range: a string An empty string indicates that all temporary objects are created only in the default tablespace of the current database. For details, see [default_tablespace](#).

Default value: empty

check_function_bodies

Parameter description: Specifies whether to enable validation of the function body string during the execution of **CREATE FUNCTION**. Verification is occasionally disabled to avoid problems, such as forward references when you restore function definitions from a dump.

Type: USERSET

Value range: Boolean

- **on** indicates that validation of the function body string is enabled during the execution of **CREATE FUNCTION**.

- **off** indicates that validation of the function body string is disabled during the execution of **CREATE FUNCTION**.

Default value: on

default_transaction_isolation

Parameter description: Specifies the default isolation level of each transaction.

Type: USERSET

Value range: enumerated values

- **READ COMMITTED:** Only committed data is read. This is the default.
- **READ UNCOMMITTED:** DWS does not support **READ UNCOMMITTED**. If **READ UNCOMMITTED** is set, **READ COMMITTED** is executed instead.
- **REPEATABLE READ:** Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE:** DWS does not support **SERIALIZABLE**. If **SERIALIZABLE** is set, **REPEATABLE READ** is executed instead.

Default value: READ COMMITTED

default_transaction_read_only_probe

Parameter description: Specifies whether to terminate the execution of special statements (e.g., statements for flushing data to disks and generating new tables or physical files) when the database is about to become read-only (disk usage reaches 90%). The CM module checks and sets the disk usage threshold. It is not advised to set this parameter. This is supported only by clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution of the special statement is terminated.
- **off** indicates that the execution of the special statement is not terminated.

Default value: off

default_transaction_deferrable

Parameter description: Specifies the default delaying state of each new transaction. It currently has no effect on read-only transactions or those running at isolation levels lower than serializable.

DWS does not support the serializable isolation level of each transaction. The parameter is insignificant.

Type: USERSET

Value range: Boolean

- **on** indicates a transaction is delayed by default.

- **off** indicates a transaction is not delayed by default.

Default value: off

session_replication_role

Parameter description: Specifies the behavior of replication-related triggers and rules for the current session.

Type: USERSET

NOTICE

Setting this parameter will discard all the cached execution plans.

Value range: enumerated values

- **origin** indicates that the system copies operations such as insert, delete, and update from the current session.
- **replica** indicates that the system copies operations such as insert, delete, and update from other places to the current session.
- **local** indicates that the system will detect the role that has logged in to the database when using the function to copy operations and will perform related operations.

Default value: origin

statement_timeout

Parameter description: If the statement execution time (starting when the server receives the command) is longer than the duration specified by the parameter, error information is displayed when you attempt to execute the statement and the statement then exits.

Type: USERSET

Value range: an integer ranging from 0 to 2147483647. The unit is ms.

Default value:

- If the current cluster is upgraded from an earlier version to 8.2.0, the value in the earlier version is inherited. The default value is **0**.
- If the cluster version 8.2.0 is newly installed, the default value is **24h**.

vacuum_freeze_min_age

Parameter description: Specifies the minimum cutoff age (in the same transaction), based on which **VACUUM** decides whether to replace transaction IDs with FrozenXID while scanning a table.

Type: USERSET

Value range: an integer from 0 to 576460752303423487.

 NOTE

Although you can set this parameter to a value ranging from **0** to **1000000000** anytime, **VACUUM** will limit the effective value to half the value of **autovacuum_freeze_max_age** by default.

Default value: 5000000000

vacuum_freeze_table_age

Parameter description: Specifies the time that VACUUM freezes tuples while scanning the whole table. **VACUUM** performs a whole-table scan if the value of the [pg_class.relfrozexid](#) column of the table has reached the specified time.

Type: USERSET

Value range: an integer from 0 to 576460752303423487.

 NOTE

Although users can set this parameter to a value ranging from **0** to **2000000000** anytime, **VACUUM** will limit the effective value to 95% of **autovacuum_freeze_max_age** by default. Therefore, a periodic manual VACUUM has a chance to run before an anti-wraparound autovacuum is launched for the table.

Default value: 15000000000

bytea_output

Parameter description: Specifies the output format for values of the bytea type.

Type: USERSET

Value range: enumerated values

- **hex** indicates the binary data is converted to the two-byte hexadecimal digit.
- **escape** indicates the traditional PostgreSQL format is used. It takes the approach of representing a binary string as a sequence of ASCII characters, while converting those bytes that cannot be represented as an ASCII character into special escape sequences.

Default value: hex

xmlbinary

Parameter description: Specifies how binary values are to be encoded in XML.

Type: USERSET

Value range: enumerated values

- base64
- hex

Default value: base64

xmloption

Parameter description: Specifies whether DOCUMENT or CONTENT is implicit when converting between XML and string values.

Type: USERSET

Value range: enumerated values

- **document** indicates an HTML document.
- **content** indicates a common string.

Default value: content

gin_pending_list_limit

Parameter description: Specifies the maximum capacity of the pending list when FASTUPDATE is enabled for GIN indexes.

Pending List is a data structure specific to GIN indexes and is used to temporarily store index update operations. When the **FASTUPDATE** parameter is enabled for a GIN index, new index items are not directly written to the main index structure, but are stored in the pending list. When conditions are met, the index items are merged into the main index in batches.

If the pending list grows larger than this maximum size, data in the list will be moved to the GIN index data structure in batches. This setting can be overridden for individual GIN indexes by modifying index storage parameters.

Type: USERSET

Value range: an integer ranging from 64 to INT_MAX. The unit is KB.

Default value: 4 MB

19.14.2 Zone and Formatting

This section describes parameters related to the time format setting.

DateStyle

Parameter description: Specifies the display format for date and time values, as well as the rules for interpreting ambiguous date input values.

This variable contains two independent components: the output format specifications (ISO, Postgres, SQL, or German) and the input/output order of year/month/day (DMY, MDY, or YMD). The two components can be set separately or together. The keywords Euro and European are synonyms for DMY; the keywords US, NonEuro, and NonEuropean are synonyms for MDY.

Type: USERSET

Value range: a string

Default value: ISO, MDY

NOTE

gs_initdb will initialize this parameter so that its value is the same as that of **lc_time**.

Suggestion: The ISO format is recommended. Postgres, SQL, and German use abbreviations for time zones, such as **EST**, **WST**, and **CST**.

IntervalStyle

Parameter description: Specifies the display format for interval values.

Type: USERSET

Value range: enumerated values

- **sql_standard** indicates that output matching SQL standards will be generated.
- **postgres** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **ISO**.
- **postgres_verbose** indicates that output matching PostgreSQL 8.4 will be generated when the **DateStyle** parameter is set to **non_ISO**.
- **iso_8601** indicates that output matching the time interval "format with designators" defined in ISO 8601 will be generated.
- **oracle** indicates the output result that matches the numtodsinterval function in the Oracle database. For details, see numtodsinterval.

NOTICE

The **IntervalStyle** parameter also affects the interpretation of ambiguous interval input.

Default value: postgres

TimeZone

Parameter description: Specifies the time zone for displaying and interpreting time stamps.

Type: USERSET

Value range: a string. You can obtain it by querying the **pg_timezone_names** view.

Default value: UTC

NOTE

gs_initdb will set a time zone value that is consistent with the system environment.

timezone_abbreviations

Parameter description: Specifies the time zone abbreviations that will be accepted by the server.

Type: USERSET

Value range: a string. You can obtain it by querying the **pg_timezone_names** view.

Default value: Default

 NOTE

Default indicates an abbreviation that works in most of the world. There are also other abbreviations, such as **Australia** and **India** that can be defined for a particular installation.

extra_float_digits

Parameter description: Specifies the number of digits displayed for floating-point values, including float4, float8, and geometric data types. The parameter value is added to the standard number of digits (FLT_DIG or DBL_DIG as appropriate).

Type: USERSET

Value range: an integer ranging from -15 to 3

 NOTE

- This parameter can be set to **3** to include partially-significant digits. It is especially useful for dumping float data that needs to be restored exactly.
- This parameter can also be set to a negative value to suppress unwanted digits.

Default value: 0

client_encoding

Parameter description: Specifies the client-side encoding type (character set).

Set this parameter as needed. Try to keep the client code and server code consistent to improve efficiency.

Type: USERSET

Value range: encoding compatible with PostgreSQL. **UTF8** indicates that the database encoding is used.

 NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.
- To use consistent encoding for communication within a cluster, you are advised to retain the default value of **client_encoding**. Modification to this parameter in the **postgresql.conf** file (by using the **gs_guc** tool, for example) does not take effect.

Default value: UTF8

Recommended value: SQL_ASCII or UTF8

lc_messages

Parameter description: Specifies the language in which messages are displayed.

Valid values depend on the current system. On some systems, this zone category does not exist. Setting this variable will still work, but there will be no effect. In addition, translated messages for the desired language may not exist. In this case, you can still see the English messages.

Type: SUSET

Value range: a string

 NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: C

lc_monetary

Parameter description: Specifies the display format of monetary values. It affects the output of functions such as `to_char`. Valid values depend on the current system.

Type: USERSET

Value range: a string

 NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: C

lc_numeric

Parameter description: Specifies the display format of numbers. It affects the output of functions such as `to_char`. Valid values depend on the current system.

Type: USERSET

Value range: a string

 NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: C

lc_time

Parameter description: Specifies the display format of time and zones. It affects the output of functions such as `to_char`. Valid values depend on the current system.

Type: USERSET

Value range: a string

 NOTE

- You can run the **locale -a** command to check and set the system-supported zone and the corresponding encoding format.
- By default, **gs_initdb** will initialize the setting of this parameter based on the current system environment. You can also run the **locale** command to check the current configuration environment.

Default value: C

default_text_search_config

Parameter description: Specifies the text search configuration.

If the specified text search configuration does not exist, an error will be reported. If the specified text search configuration is deleted, set **default_text_search_config** again. Otherwise, an error will be reported, indicating incorrect configuration.

- The text search configuration is used by text search functions that do not have an explicit argument specifying the configuration.
- When a configuration file matching the environment is determined, **gs_initdb** will initialize the configuration file with a setting that corresponds to the environment.

Type: USERSET

Value range: a string

 NOTE

DWS supports the following two configurations: **pg_catalog.english** and **pg_catalog.simple**.

Default value: **pg_catalog.english**

19.14.3 Other Default Parameters

This section describes the default database loading parameters of the database system.

dynamic_library_path

Parameter description: Specifies the path for saving the shared database files that are dynamically loaded for data searching. When a dynamically loaded module needs to be opened and the file name specified in the **CREATE FUNCTION** or **LOAD** command does not have a directory component, the system will search this path for the required file.

The value of **dynamic_library_path** must be a list of absolute paths separated by colons (:) or by semi-colons (;) on the Windows OS. The special variable **\$libdir** in the beginning of a path will be replaced with the module installation directory provided by DWS. Example:

```
dynamic_library_path = '/usr/local/lib/postgresql:/opt/testgs/lib:$libdir'
```

Type: SUSET

Value range: a string

 NOTE

If the value of this parameter is set to an empty character string, the automatic path search is turned off.

Default value: \$libdir

gin_fuzzy_search_limit

Parameter description: Specifies the upper limit of the size of the set returned by GIN indexes.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX. The value **0** indicates no limit.

Default value: 0

19.15 Lock Management

In DWS, concurrent transactions may cause single-node deadlocks or distributed deadlocks due to resource competition. This section describes parameters used for managing transaction lock mechanisms.

deadlock_timeout

Parameter description: Specifies the time, in milliseconds, to wait on a lock before checking whether there is a deadlock condition. When the applied lock exceeds the preset value, the system will check whether a deadlock occurs.

- The check for deadlock is relatively expensive. Therefore, the server does not check it when waiting for a lock every time. Deadlocks do not frequently occur when the system is running. Therefore, the system just needs to wait on the lock for a while before checking for a deadlock. Increasing this value reduces the time wasted in needless deadlock checks, but slows down reporting of real deadlock errors. On a heavily loaded server, you may need to raise it. The value you have set needs to exceed the transaction time. By doing this, the possibility that a lock will be released before the waiter decides to check for deadlocks will be reduced.
- When [log_lock_waits](#) is set, this parameter also determines the duration you need to wait before a log message about the lock wait is issued. If you are trying to investigate locking delays, you need to set this parameter to a value smaller than normal **deadlock_timeout**.

Type: SUSET

Value range: an integer ranging from 1 to 2147483647. The unit is millisecond (ms).

Default value: 1s

ddl_lock_timeout

Parameter description: Indicates the number of seconds a DDL command should wait for the locks to become available. If the time spent in waiting for a lock exceeds the specified time, an error is reported. This parameter is supported only by clusters of version 8.1.3.200 or later.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

- If the value of this parameter is 0, this parameter does not take effect.
- If the value of this parameter is greater than 0, the lock wait time of DDL statements is the value of this parameter, and the lock wait time of other locks is the value of **lockwait_timeout**.

Default value: 0

NOTE

This parameter has a higher priority than **lockwait_timeout** and takes effect only for **AccessExclusiveLock**.

ddl_select_concurrent_mode

Parameter description: Specifies the concurrency mode of DDL and **SELECT** statements. This parameter is supported only by clusters of version 8.1.3.320, 8.2.1, or later.

Type: SUSET

Value range: a string

- **none:** DDL and select statements cannot be executed concurrently. Waiting statements are in the lock wait state.
- **truncate:** When the **TRUNCATE** statement is blocked by the **SELECT** statement, the **SELECT** statement is interrupted and the **TRUNCATE** statement is executed first.
- **exchange:** When the **EXCHANGE** statement is blocked by the **SELECT** statement, the **SELECT** statement is interrupted and the **EXCHANGE** statement is executed first.
- **vacuum_full:** When the **vacuum_full** statement is blocked by the **SELECT** statement, the **SELECT** statement is interrupted and the **vacuum_full** statement is executed first. This is supported only by clusters of version 9.1.0.200 or later.
- **insert_overwrite:** When the **insert_overwrite** statement is blocked by the **SELECT** statement, the **SELECT** statement is interrupted and the **insert_overwrite** statement is executed first. This is supported only by clusters of version 9.1.0.200 or later.

Default value: none

 NOTE

- To reserve time for the **SELECT** statement to respond to signals, if the value of **ddl_lock_timeout** is less than 1 second in the current version, 1 second is used.
- Concurrency is not supported when there are conflicts with locks of higher levels (more than one level). For example, **autoanalyze** is triggered by **SELECT** when **autoanalyze_mode** is set to **normal**.
- This parameter allows for **SELECT** statements in either a single statement or a transaction block. However, in other versions, it only supports **SELECT** statements in a single statement. For concurrent **SELECT** operations in a single statement or transaction block, learn more information in the description of parameter [enable_cancel_select_in_txnblock](#).
- Values other than **none** can be used together. For example, if this parameter is set to **truncate**, **exchange**, the **TRUNCATE** and **EXCHANGE** statements are blocked by the **SELECT** statement. The **SELECT** statement is interrupted and executed first.

enable_cancel_select_in_txnblock

Parameter description: Specifies whether the **SELECT** statement in a transaction block can be interrupted.

Type: USERSET

Value range: Boolean

- **on** indicates that the select operation in the transaction block can be interrupted.
- **off** indicates that the select operation in the transaction block cannot be interrupted.

Default value: on

 NOTE

- This parameter is supported by 8.2.1.236 and later versions in 8.2.1.2 xx clusters. This parameter is supported by clusters of version 9.1.0.200 or later.
- This parameter controls whether the **SELECT** statement in a transaction block can be interrupted by the DDL operation specified in **ddl_select_concurrent_mode**.
- The **ddl_select_concurrent_mode** parameter controls DDL statements such as **TRUNCATE** and **EXCHANGE**, and the **enable_cancel_select_in_txnblock** parameter controls **SELECT** statements.

lockwait_timeout

Parameter description: Specifies the longest time to wait before a single lock times out. If the time you wait before acquiring a lock exceeds the specified time, an error is reported.

Type: SUSERSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

Default value: 20 min

update_lockwait_timeout

Parameter description: sets the maximum duration that a lock waits for concurrent updates on a row to complete when the concurrent update feature is

enabled. If the time you wait before acquiring a lock exceeds the specified time, an error is reported.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX. The unit is millisecond (ms).

Default value: 2min

partition_lock_upgrade_timeout

Parameter description: Specifies the time to wait before the attempt of a lock upgrade from ExclusiveLock to AccessExclusiveLock times out on partitions.

- When you do MERGE PARTITION and CLUSTER PARTITION on a partitioned table, temporary tables are used for data rearrangement and file exchange. To concurrently perform as many operations as possible on the partitions, ExclusiveLock is acquired for the partitions during data rearrangement and AccessExclusiveLock is acquired during file exchange.
- Generally, a partition waits until it acquires a lock, or a timeout occurs if the partition waits for a period of time longer than specified by the [lockwait_timeout](#) parameter.
- When doing MERGE PARTITION or CLUSTER PARTITION on a partitioned table, you need to acquire AccessExclusiveLock during file exchange. If the lock fails to be acquired, the acquisition is retried in 50 ms. This parameter specifies the time to wait before the lock acquisition attempt times out.
- If this parameter is set to -1, the lock upgrade never times out. The lock upgrade is continuously retried until it succeeds.

Type: USERSET

Value range: an integer ranging from -1 to 3000, in seconds

Default value: 1800

enable_release_scan_lock

Parameter description: Specifies whether a SELECT statement releases a level-1 lock after the statement execution is complete. This parameter reduces DDL conflicts with SELECT locks within transaction blocks. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that DDL operations will be blocked to wait for the release of cluster locks. The SELECT statement releases the level-1 lock after it finishes, not when the transaction commits.
- **off** indicates that DDL operations will not be blocked.

Default value: off

vacuum_full_interruptible

Parameter description: Controls the behavior that the **VACUUM FULL** statement gives a lock to other statements. This is supported only by clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that DDL operations will be blocked to wait for the release of cluster locks. When **VACUUM FULL** blocks other statements, it interrupts the execution and gives the lock to other statements.
- **off** indicates that DDL operations will not be blocked. When **VACUUM FULL** blocks other statements, it does not interrupt the execution. Other statements can be executed only after **VACUUM FULL** has completed and released the lock.

Default value: off

19.16 Version and Platform Compatibility

19.16.1 Compatibility with Earlier Versions

DWS provides parameter controls for the downward compatibility and external compatibility features of the database. The backward compatibility of the database system can provide support for old versions of database applications. The parameters introduced in this section mainly control the backward compatibility of the database.

array_nulls

Parameter description: Determines whether the array input parser recognizes unquoted NULL as a null array element.

Type: USERSET

Value range: Boolean

- **on** indicates that null values can be entered in arrays.
- **off** indicates backward compatibility with the old behavior. Arrays containing **NULL** values can still be created when this parameter is set to **off**.

Default value: on

backslash_quote

Parameter description: Determines whether a single quotation mark can be represented by \' in a string text.

Type: USERSET

NOTICE

When the string text meets the SQL standards, \ has no other meanings. This parameter only affects the handling of non-standard-conforming string texts, including escape string syntax (E'...').

Value range: enumerated values

- **on** indicates that the use of \ is always allowed.
- **off** indicates that the use of \ is rejected.
- **safe_encoding** indicates that the use of \ is allowed only when client encoding does not allow ASCII \ within a multibyte character.

Default value: **safe_encoding**

default_with_oids

Parameter description: Determines whether **CREATE TABLE** and **CREATE TABLE AS** include an **OID** field in newly-created tables if neither **WITH OIDS** nor **WITHOUT OIDS** is specified. It also determines whether OIDs will be included in tables created by **SELECT INTO**.

It is not recommended that OIDs be used in user tables. Therefore, this parameter is set to **off** by default. When OIDs are required for a particular table, **WITH OIDS** needs to be specified during the table creation.

Type: USERSET

Value range: Boolean

- **on** indicates **CREATE TABLE** and **CREATE TABLE AS** can include an **OID** field in newly-created tables.
- **off** indicates **CREATE TABLE** and **CREATE TABLE AS** cannot include any **OID** field in newly-created tables.

Default value: **off**

enable_parquet_integer_decimal

Parameter description: Specifies whether to use the **INT64** or **FIXED_LEN_BYTE_ARRAY** type for storage when the **decimal** (p,s) type is written in the Parquet foreign tables and p is less than or equal to 18.

Type: USERSET

Value range: Boolean

- **on:** The **INT64** type is used for storage.
- **off:** The **FIXED_LEN_BYTE_ARRAY** is used for storage.

Default value: **off**

 NOTE

- In cluster version 9.1.0, only 9.1.0.222 and later versions support this parameter. To be compatible with the Hive behavior (Hive uses **FIXED_LEN_BYTE_ARRAY**), the default value of this parameter is **FIXED_LEN_BYTE_ARRAY** and historical behavior compatibility is retained.
- In cluster versions that do not support this parameter, the Parquet foreign tables in DWS store the $p \leq 18$ decimal as the **INT64** type by default.
- **INT64** or **FIXED_LEN_BYTE_ARRAY** indicates the type defined in Parquet. Different external engines may select different types.

escape_string_warning

Parameter description: Specifies a warning on directly using a backslash (\) as an escape in an ordinary character string.

- Applications that wish to use a backslash (\) as an escape need to be modified to use escape string syntax (E'...'). This is because the default behavior of ordinary character strings is now to treat the backslash as an ordinary character in each SQL standard.
- This variable can be enabled to help locate codes that need to be changed.

Type: USERSET

Value range: Boolean

Default value: on

lo_compat_privileges

Parameter description: Determines whether to enable backward compatibility for the privilege check of large objects.

Type: SUSET

Value range: Boolean

on indicates that the privilege check is disabled when users read or modify large objects. This setting is compatible with versions earlier than PostgreSQL 9.0.

Default value: off

quote_all_identifiers

Parameter description: When the database generates SQL, this parameter forcibly quotes all identifiers even if they are not keywords. This will affect the output of EXPLAIN as well as the results of functions, such as pg_get_viewdef. For details, see the **--quote-all-identifiers** option in section "gs_dump" in the *Data Warehouse Service (DWS) Tool Guide*.

Type: USERSET

Value range: Boolean

- **on** indicates the forcible quotation function is enabled.
- **off** indicates the forcible quotation function is disabled.

Default value: off

sql_inheritance

Parameter description: Determines whether to inherit semantics.

Type: USERSET

Value range: Boolean

off indicates that child tables cannot be accessed by various commands. That is, an **ONLY** keyword is used by default. This setting is compatible with versions earlier than PostgreSQL 7.1.

Default value: on

standard_conforming_strings

Parameter description: Determines whether ordinary string texts ('...') treat backslashes as ordinary texts as specified in the SQL standard.

- Applications can check this parameter to determine how string texts will be processed.
- It is recommended that characters be escaped by using the escape string syntax (E'...').

Type: USERSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: on

synchronize_seqscans

Parameter description: Controls sequential scans of tables to synchronize with each other. Concurrent scans read the same data block about at the same time and share the I/O workload.

Type: USERSET

Value range: Boolean

- **on** indicates that a scan may start in the middle of the table and then "wrap around" the end to cover all rows to synchronize with the activity of scans already in progress. This may result in unpredictable changes in the row ordering returned by queries that have no **ORDER BY** clause.
- **off** indicates that the scan always starts from the table heading.

Default value: on

enable_beta_features

Parameter description: Controls whether certain limited features, such as GDS table join, are available. These features are not explicitly prohibited in earlier versions, but are not recommended due to their limitations in certain scenarios.

Type: USERSET

Value range: Boolean

- **on** indicates that the features are enabled and forward compatible, but may incur errors in certain scenarios.
- **off** indicates that the features are disabled.

Default value: off

19.16.2 Platform and Client Compatibility

Database systems are widely used across many platforms, and their external compatibility offers a great deal of convenience.

transform_null_equals

Parameter description: Determines whether expressions of the form `expr = NULL` (or `NULL = expr`) are treated as `expr IS NULL`. They return true if `expr` evaluates to **NULL**, and false otherwise.

- The correct SQL-standard-compliant behavior of `expr = NULL` is to always return null (unknown).
- Filtered forms in Microsoft Access generate queries that appear to use `expr = NULL` to test for null values. If you turn this option on, you can use this interface to access the database.

Type: USERSET

Value range: Boolean

- **on** indicates expressions of the form `expr = NULL` (or `NULL = expr`) are treated as `expr IS NULL`.
- **off** indicates `expr = NULL` always returns **NULL**.

Default value: off

NOTE

New users are always confused about the semantics of expressions involving **NULL** values. Therefore, **off** is used as the default value.

td_compatible_truncation

Parameter description: Determines whether to enable features compatible with a Teradata database. You can set this parameter to **on** when connecting to a database compatible with the Teradata database, so that when you perform the INSERT operation, overlong strings are truncated based on the allowed maximum length before being inserted into char- and varchar-type columns in the target table. This ensures all data is inserted into the target table without errors reported.

 NOTE

- The string truncation function cannot be used if the **INSERT** statement includes a foreign table.
- If inserting multi-byte character data (such as Chinese characters) to database with the character set byte encoding (SQL_ASCII, LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF8, which has no character data crossing the truncation position.

Type: USERSET

Value range: Boolean

- **on** indicates overlong strings are truncated.
- **off** indicates overlong strings are not truncated.

Default value: off

behavior_compat_options

Parameter description: Specifies the database compatibility behavior, which consists of multiple items separated by commas (,). Compatibility configurations are applied according to the database type (such as Oracle/Teradata/MySQL). For details, see [Table 19-2](#).

Type: USERSET

Value range: a string

Default value: In upgrade scenarios, the default value of this parameter is the same as that in the cluster before the upgrade. When a new cluster is installed, the default value of this parameter is

check_function_conflicts,check_function_shippable,unsupported_set_function_case to prevent serious issues caused by incorrect function attributes that users define.

 NOTE

- Currently, only items in [Table 19-2](#) are supported.
- Multiple items are separated by commas (,), for example, **set behavior_compat_options='end_month_calculate,display_leading_zero';**
- **strict_concat_functions** and **strict_text_concat_td** are mutually exclusive.
- You are not advised to set **behavior_compat_options** to **'return_null_string'** in Oracle compatibility mode. If this option is set, do not insert query results into tables.

Table 19-2 Compatibility configuration items

| Configuratio
n Item | Behavior | Appl
icabl
e Com
pati
bilit
y Mod
e |
|---------------------------|---|---|
| display_leadi
ng_zero | <p>Specifies how floating point numbers are displayed.</p> <ul style="list-style-type: none">• If this item is not specified, decimal numbers between -1 and 0, and between 0 and 1, do not display the leading zero before the decimal point. For example, 0.25 is displayed as .25.• If this item is specified, decimal numbers between -1 and 0, and between 0 and 1, display the leading zero before the decimal point. For example, 0.25 is displayed as 0.25. | ORA
TD |
| end_month_c
alculate | <p>Specifies the calculation logic of the add_months function.</p> <p>Assuming that the two parameters of the add_months function are param1 and param2, and the sum of the months of param1 and param2 is result:</p> <ul style="list-style-type: none">• If this item is not specified, and the Day of param1 indicates the last day of a month shorter than result, the Day in the calculation result will equal that in param1. For example: <pre>select add_months('2018-02-28',3) from dual;
add_months

2018-05-28 00:00:00
(1 row)</pre> <ul style="list-style-type: none">• If this item is specified, and the Day of param1 indicates the last day of a month shorter than result, the Day in the calculation result will equal that in result. For example: <pre>select add_months('2018-02-28',3) from dual;
add_months

2018-05-31 00:00:00
(1 row)</pre> | ORA
TD |
| compat_anal
yze_sample | <p>Specifies the sampling behavior of the ANALYZE operation.</p> <p>If this item is specified, the sample collected by the ANALYZE operation will be limited to around 30,000 records, controlling CN memory consumption and maintaining the stability of ANALYZE.</p> | ORA
TD
MyS
QL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------------|---|-------------------------------|
| bind_schema_tablespace | Binds a schema with the tablespace with the same name.

If a tablespace name is the same as <i>schema_name</i> , default_tablespace will also be set to <i>schema_name</i> if search_path is set to <i>schema_name</i> . | ORA
TD
MySQL |
| bind_procedure_searchpath | Specifies the search path of the database object for which no schema name is specified.

If no schema name is specified for a stored procedure, the search is performed in the schema the stored procedure belongs to.

If the stored procedure is not found, the following operations are performed: <ul style="list-style-type: none"> • If this item is not specified, the system reports an error and exits. • If this item is specified, the search continues based on the settings of search_path. If the issue persists, the system reports an error and exits. | ORA
TD
MySQL |
| correct_to_number | Controls the compatibility of the to_number() result.

If this item is specified, the result of the to_number() function is the same as that of PG11. Otherwise, the result is the same as that of Oracle. | ORA |
| unbind_divide_bound | Controls the range check on the result of integer division. <ul style="list-style-type: none"> • If this item is not specified, the division result is checked. If the result is out of the range, an error is reported. In the following example, an out-of-range error is reported because the value of INT_MIN/(-1) is greater than the value of INT_MAX.

 <pre>SELECT (-2147483648)::int / (-1)::int; ERROR: integer out of range</pre> • If this item is specified, the range of the division result does not need to be checked. In the following example, INT_MIN/(-1) can be used to obtain the output result INT_MAX+1.

 <pre>SELECT (-2147483648)::int / (-1)::int; ?column? ----- 2147483648 (1 row)</pre> | ORA
TD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------------|---|-------------------------------|
| merge_update_multi | Specifies whether to perform an update when MERGE INTO is executed to match multiple rows.

If this item is specified, no error is reported when multiple rows are matched. Otherwise, an error is reported (same as Oracle). | ORATD |
| disable_row_update_multi | Specifies whether to perform an update when multiple rows of a row-store table are matched.

If this item is specified, an error is reported when multiple rows are matched. Otherwise, multiple rows can be matched and updated by default. | ORATD |
| return_null_string | Specifies how to display the empty result (empty string ") of the lpad() , rpadd() , repeat() , regexp_split_to_table() , and split_part() functions. <ul style="list-style-type: none">If this item is not specified, the empty string is displayed as NULL.
<pre>select length(lpad('123',0,'*')) from dual;
length

(1 row)</pre>If this item is specified, the empty string is displayed as single quotation marks (").
<pre>select length(lpad('123',0,'*')) from dual;
length

0
(1 row)</pre> | ORA |
| compat_concat_variadic | Specifies the compatibility of variadic results of the concat() and concat_ws() functions.

If this item is specified and a concat function has a parameter of the variadic type, different result formats in Oracle and Teradata are retained. If this item is not specified and a concat function has a parameter of the variadic type, the result format of Oracle is retained for both Oracle and Teradata. | ORATD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------------------|---|-------------------------------|
| convert_string_digit_to_numeric | <p>Specifies the type casting priority for binary BOOL operations on the CHAR type and INT type.</p> <ul style="list-style-type: none">• If this item is not specified, the type casting priority is the same as that of PG9.6.• After this item is configured, all binary BOOL operations of the CHAR type and INT type are forcibly converted to the NUMERIC type for computation. After this configuration item is set, the CHAR types that are affected include BPCHAR, VARCHAR, NVARCHAR2, and TEXT, and the INT types that are affected include INT1, INT2, INT4, and INT8. <p>CAUTION
This configuration item is valid only for binary BOOL operation, for example, INT2>TEXT and INT4=BPCHAR. Non-BOOL operation is not affected. This configuration item does not support conversion of UNKNOWN operations such as INT>'1.1'. After this configuration item is enabled, all BOOL operations of the CHAR and INT types are preferentially converted to the NUMERIC type for computation, which affects the computation performance of the database. When the JOIN column is a combination of affected types, the execution plan is affected.</p> | ORAD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------------|--|-------------------------------|
| check_function_conflicts | <p>Controls the check of the custom plpgsql/SQL function attributes.</p> <ul style="list-style-type: none">• If this parameter is not specified, the IMMUTABLE/STABLE/VOLATILE attributes of a custom function are not checked.• If this parameter is specified, the IMMUTABLE attribute of a custom function is checked. If the function contains a table or the STABLE/VOLATILE function, an error is reported during the function execution. In a custom function, a table or the STABLE/VOLATILE function conflicts with the IMMUTABLE attribute, thus function behaviors are not IMMUTABLE in this case. <p>For example, when this parameter is specified, an error is reported in the following scenarios:</p> <pre>CREATE OR replace FUNCTION sql_immutable (INTEGER) RETURNS INTEGER AS 'SELECT a+\$1 from shipping_schema.t4 where a=1;' LANGUAGE SQL IMMUTABLE RETURNS NULL ON NULL INPUT; select sql_immutable(1); ERROR: IMMUTABLE function cannot contain SQL statements with relation or Non-IMMUTABLE function. CONTEXT: SQL function "sql_immutable" during startup referenced column: sql_immutable</pre> | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------|--|-------------------------------|
| varray_verification | <p>Indicates whether to verify the array length and array type length. This parameter is compatible with DWS of versions earlier than 8.1.0.</p> <p>If this parameter is specified, the array length and array type length are not verified.</p> <p>Scenario 1</p> <pre>CREATE OR REPLACE PROCEDURE varray_verification AS TYPE org_varray_type IS varray(5) OF VARCHAR2(2); v_org_varray org_varray_type; BEGIN v_org_varray(1) := '111'; --If the value exceeds the limit of VARCHAR2(2), the setting will be consistent with that in the historical version and no verification is performed after configuring this option. END; /</pre> <p>Scenario 2</p> <pre>CREATE OR REPLACE PROCEDURE varray_verification_i3_1 AS TYPE org_varray_type IS varray(2) OF NUMBER(2); v_org_varray org_varray_type; BEGIN v_org_varray(3) := 1; --If the value exceeds the limit of varray(2) specified for array length, the setting will be consistent with that in the historical version and no verification is performed after configuring this option. END; /</pre> | ORATD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------|---|-------------------------------|
| strict_concat_functions | <p>Indicates whether the textanycat() and anytextcat() functions are compatible with the return value if there are null parameters. This parameter and strict_text_concat_td are mutually exclusive.</p> <p>In MySQL-compatible mode, this parameter has no impact.</p> <ul style="list-style-type: none">• If this configuration item is not specified, the returned values of the textanycat() and anytextcat() functions are the same as those in the Oracle database.• When this configuration item is specified, if there are null parameters in the textanycat() and anytextcat() functions, the returned value is also null. Different result formats in Oracle and Teradata are retained. <p>If this configuration item is not specified, the returned values of the textanycat() and anytextcat() functions are the same as those in the Oracle database.</p> <pre>SELECT textanycat('gauss', cast(NULL as BOOLEAN)); textanycat ----- gauss (1 row)</pre> <p>SELECT 'gauss' cast(NULL as BOOLEAN); -- In this case, the operator is converted to the textanycat function.</p> <pre>?column? ----- gauss (1 row)</pre> <p>When setting this configuration item, retain the results that are different from those in Oracle and Teradata:</p> <pre>SELECT textanycat('gauss', cast(NULL as BOOLEAN)); textanycat ----- (1 row)</pre> <pre>SELECT 'gauss' cast(NULL as BOOLEAN); -- In this case, the operator is converted to the textanycat function. ?column? ----- (1 row)</pre> | ORAD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-----------------------|--|-------------------------------|
| strict_text_concat_td | <p>In Teradata compatible mode, whether the textcat(), textanycat() and anytextcat() functions are compatible with the return value if there are null parameters. This parameter and strict_concat_functions are mutually exclusive.</p> <ul style="list-style-type: none">• If this parameter is not specified, the return values of the textcat(), textanycat(), and anytextcat() functions in Teradata-compatible mode are the same as those in DWS.• When this parameter is specified, if the textcat(), textanycat(), and anytextcat() functions contain any null parameter values, the return value is null in Teradata-compatible mode. <p>If this parameter is not specified, the return values of the textcat(), textanycat(), and anytextcat() functions are the same as those in DWS.</p> <pre>td_compatibility_db=# SELECT textcat('abc', NULL); textcat ----- abc (1 row) td_compatibility_db=# SELECT 'abc' NULL; -- In this case, the operator is converted to the textcat() function. ?column? ----- abc (1 row)</pre> <p>When this parameter is specified, NULL is returned if any of the textcat(), textanycat(), and anytextcat() functions returns a null value.</p> <pre>td_compatibility_db=# SELECT textcat('abc', NULL); textcat ----- (1 row) td_compatibility_db=# SELECT 'abc' NULL; ?column? ----- (1 row)</pre> | TD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------------|--|-------------------------------|
| compat_display_ref_table | <p>Sets the column display format in the view.</p> <ul style="list-style-type: none">If this parameter is not specified, the prefix is used by default, in the tab.col format.Specify this parameter to the same original definition. It is displayed only when the original definition contains a prefix. <pre>SET behavior_compat_options='compat_display_ref_table'; CREATE OR REPLACE VIEW viewtest2 AS SELECT a.c1, c2, a.c3, 0 AS c4 FROM viewtest_tbl a; SELECT pg_get_viewdef('viewtest2'); pg_get_viewdef ----- SELECT a.c1, c2, a.c3, 0 AS c4 FROM viewtest_tbl a; (1 row)</pre> | ORATD |
| para_support_set_func | <p>Whether the input parameters of the COALESCE(), NVL(), GREATEST(), and LEAST() functions in a column-store table support multiple result set expressions.</p> <ul style="list-style-type: none">If this item is not specified and the input parameter contains multiple result set expressions, an error is reported, indicating that the function is not supported. <pre>SELECT COALESCE(regexp_split_to_table(c3,'#'), regexp_split_to_table(c3,'#')) FROM regexp_ext2_tb1 ORDER BY 1 LIMIT 5; ERROR: set-valued function called in context that cannot accept a set</pre> <ul style="list-style-type: none">When this configuration item is specified, the function input parameter can contain multiple result set expressions. <pre>SELECT COALESCE(regexp_split_to_table(c3,'#'), regexp_split_to_table(c3,'#')) FROM regexp_ext2_tb1 ORDER BY 1 LIMIT 5; coalesce ----- a a a a a (5 rows)</pre> | ORATD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|----------------------------------|--|-------------------------------|
| disable_select_truncate_parallel | <p>Controls the DDL lock level such as TRUNCATE in a partitioned table.</p> <ul style="list-style-type: none">If this item is specified, the concurrent execution of TRUNCATE and DML operations (such as SELECT) on different partitions is forbidden, and the fast query shipping (FQS) of the SELECT operation on the partitioned table is allowed. You can set this parameter in the OLTP database, where there are many simple queries on partitioned tables, and there is no requirement for concurrent TRUNCATE and DML operations on different partitions.If this item is not specified, SELECT and TRUNCATE operations can be concurrently performed on different partitions in a partitioned table, and the FQS of the partitioned table is disabled to avoid possible inconsistency. | ORAD
MySQL |
| bpchar_text_without_trimm | <p>In Teradata-compatible mode, controls the space to be retained on the right during the character conversion from bpchar to text. If the actual length is less than the length specified by bpchar, spaces are added to the value to be compatible with the Teradata style of the bpchar string.</p> <p>Currently, ignoring spaces at the end of a string for comparison is not supported. If the concatenated string contains spaces at the end, the comparison is space-sensitive.</p> <p>The following is an example:</p> <pre>td_compatibility_db=# select length('a'::char(10)::text); length ----- 10 (1 row) td_compatibility_db=# select length('a' 'a'::char(10)); length ----- 11 (1 row)</pre> | TD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|------------------------------|--|-------------------------------|
| convert_empty_str_to_null_td | <p>In Teradata-compatible mode, controls the to_date, to_timestamp, and to_number type conversion functions to return null when they encounter empty strings, and controls the format of the return value when the to_char function encounters an input parameter of the date type.</p> <p>Example:</p> <p>If this parameter is not specified:</p> <pre>td_compatibility_db=# select to_number(""); to_number ----- 0 (1 row)</pre> <pre>td_compatibility_db=# select to_date(""); ERROR: the format is not correct DETAIL: invalid date length "0", must between 8 and 10. CONTEXT: referenced column: to_date</pre> <pre>td_compatibility_db=# select to_timestamp(""); to_timestamp ----- 0001-01-01 00:00:00 BC (1 row)</pre> <pre>td_compatibility_db=# select to_char(date '2020-11-16'); to_char ----- 2020-11-16 00:00:00+08 (1 row)</pre> <p>If this parameter is specified, and parameters of to_number, to_date, and to_timestamp functions contain empty strings:</p> <pre>td_compatibility_db=# select to_number(""); to_number ----- (1 row)</pre> <pre>td_compatibility_db=# select to_date(""); to_date ----- (1 row)</pre> <pre>td_compatibility_db=# select to_timestamp(""); to_timestamp ----- (1 row)</pre> | TD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------|--|-------------------------------|
| | <pre>td_compatibility_db=# select to_char(date '2020-11-16'); to_char ----- 2020/11/16 (1 row)</pre> | |
| disable_case_specific | <p>Determines whether to ignore case sensitivity during character type match. This parameter is valid only in Teradata-compatible mode.</p> <ul style="list-style-type: none"> If this item is not specified, characters are case-sensitive during character type match. If this item is specified, characters are case-insensitive during character type match. <p>After being specified, this item will affect five character types (CHAR, TEXT, BPCHAR, VARCHAR, and NVARCHAR), 12 operators (<, >, =, >=, <=, !=, <>, !=, like, not like, in, and not in), and expressions case when and decode.</p> <p>CAUTION
After this item is enabled, the UPPER function is added before the character type, which affects the estimation logic. Therefore, an enhanced estimation model is required. (Suggested settings: cost_param = 16, cost_model_version = 1, join_num_distinct = -20, and qual_num_distinct = 200)</p> | TD |
| enable_interval_to_text | <p>Controls the implicit conversion from the interval type to the text type.</p> <ul style="list-style-type: none"> When this option is enabled, the implicit conversion from the interval type to the text type is supported.
 <pre>SELECT TO_DATE('20200923', 'yyyymmdd') - TO_DATE('20200920', 'yyyymmdd') = '3'::text; ?column? ----- f (1 row)</pre> When this option is disabled, the implicit conversion from the interval type to the text type is not supported.
 <pre>SELECT TO_DATE('20200923', 'yyyymmdd') - TO_DATE('20200920', 'yyyymmdd') = '3'::text; ?column? ----- t (1 row)</pre> | ORA
TD
MyS
QL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|----------------------------------|--|-------------------------------|
| case_insensitive | <p>In MySQL-compatible mode, configure this parameter to specify the case-insensitive input parameters of the locate, strpos, and instr string functions.</p> <p>Currently, this parameter is not configured by default. That is, the input parameter is case-sensitive.</p> <p>Example:</p> <ul style="list-style-type: none">If this parameter is not configured, the input parameter is case-sensitive.
<pre>mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');
locate

0
(1 row)</pre>If this parameter is configured, the input parameter is case-insensitive.
<pre>mysql_compatibility_db=# SELECT LOCATE('sub', 'Substr');
locate

1
(1 row)</pre> | MySQL |
| inherit_not_null_strict_function | <p>Controls the original strict attribute of a function. A function with one parameter can transfer the NOT NULL attribute. func(x) is used as an example. If func() is the strict attribute and x contains the NOT NULL constraint, func(x) also contains the NOT NULL constraint.</p> <p>The compatible configuration item is effective in some optimization scenarios, for example, NOT IN and COUNT(DISTINCT) optimization. However, the optimization results may be incorrect in specific scenarios.</p> <p>Currently, this parameter is not configured by default to ensure that the result is correct. However, the performance may be rolled back. If an error occurs, you can set this parameter to roll back to the historical version.</p> | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---|---|-------------------------------|
| disable_compatibility_minmax_expr_mysql | <p>Specifies the method for processing the input parameter null in the greatest/least expression in MySQL-compatible mode.</p> <p>You can configure this parameter to roll back to a historical version.</p> <ul style="list-style-type: none">If this parameter is not configured and the input parameter is null, null is returned.
mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null);
greatest least
-----+-----

(1 row)If this parameter is configured, the maximum or minimum value of non-null parameters is returned.
mysql_compatibility_db=# SELECT greatest(1, 2, null), least(1, 2, null);
greatest least
-----+-----
 2 1
(1 row) | MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|------------------------------------|---|-------------------------------|
| disable_compatibility_substr_mysql | <p>Specifies the behavior of the substr/substring function when the start position pos is ≤ 0 in MySQL-compatible mode.</p> <p>You can configure this parameter to roll back to a historical version.</p> <ul style="list-style-type: none">If this parameter is not configured, that is, an empty string is returned when pos = 0. When pos < 0, TRUNCATE starts from the last pos character on.
<pre>mysql_compatibility_db=# SELECT substr('helloworld',0); substr ----- (1 row) mysql_compatibility_db=# SELECT substring('helloworld',0),substring('helloworld',-2,4); substring substring -----+----- ld (1 row)</pre>If this parameter is configured and pos is ≤ 0, characters are truncated from the left.
<pre>mysql_compatibility_db=# SELECT substr('helloworld',0); substr ----- helloworld (1 row) mysql_compatibility_db=# SELECT substring('helloworld',0),substring('helloworld',-2,4); substring substring -----+----- helloworld h (1 row)</pre> | MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|----------------------------------|--|-------------------------------|
| disable_compatibility_trim_mysql | <p>Specifies the method for processing the input parameter in the trim/ltrim/rtrim function in MySQL-compatible mode.</p> <p>You can configure this parameter to roll back to a historical version.</p> <ul style="list-style-type: none">If this parameter is not configured, the entire substring is matched.
<pre>mysql_compatibility_db=# SELECT trim('{}{name}{}','{}'),trim('xyznamezyx','xyz'); btrim btrim -----+----- {name} namezyx (1 row)</pre>If this parameter is configured, a single character in the character set is matched.
<pre>mysql_compatibility_db=# SELECT trim('{}{name}{}','{}'),trim('xyznamezyx','xyz'); btrim btrim -----+----- name name (1 row)</pre> | MySQL |
| light_object_mtime | <p>Specifies whether the mtime column in the pg_object system catalog records object operations.</p> <ul style="list-style-type: none">If this parameter is configured, the GRANT, REVOKE, and TRUNCATE operations are not recorded by mtime, that is, the mtime column is not updated.If this parameter is not configured (by default), the ALTER, COMMENT, GRANT, REVOKE, and TRUNCATE operations are recorded by mtime, that is, the mtime column is updated. | ORAD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-----------------------------|---|-------------------------------|
| disable_including_all_mysql | <p>In MySQL-compatible mode, this parameter controls whether the CREATE TABLE...LIKE syntax is INCLUDING_ALL.</p> <p>By default, this parameter is not set. That is, in MySQL compatibility mode, CREATE TABLE... LIKE syntax is INCLUDING_ALL.</p> <p>You can configure this parameter to roll back to a historical version.</p> <ul style="list-style-type: none"> If this parameter is not set, in MySQL-compatible mode, the CREATE TABLE... LIKE syntax is in INCLUDING_ALL. <pre>mysql_compatibility_db=# CREATE TABLE mysql_like(id int, name varchar(10), score int) DISTRIBUTE BY hash(id) COMMENT 'mysql_like'; CREATE TABLE mysql_compatibility_db=# CREATE INDEX index_like ON mysql_like(name); CREATE INDEX mysql_compatibility_db=# \d+ mysql_like; Table "public.mysql_like" Column Type Modifiers Storage Stats target Description -----+-----+-----+-----+-----+ id integer plain name character varying(10) extended score integer plain Indexes: "index_like" btree (name) TABLESPACE pg_default Has OIDs: no Distribute By: HASH(id) Location Nodes: ALL DATANODES Options: orientation=row, compression=no mysql_compatibility_db=# CREATE TABLE copy_like like mysql_like; CREATE TABLE mysql_compatibility_db=# \d+ copy_like; Table "public.copy_like" Column Type Modifiers Storage Stats target Description -----+-----+-----+-----+ id integer plain name character varying(10) extended score integer plain Indexes: "copy_like_name_idx" btree (name) TABLESPACE pg_default Has OIDs: no Distribute By: HASH(id) Location Nodes: ALL DATANODES Options: orientation=row, compression=no</pre> | MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------|--|-------------------------------|
| | <ul style="list-style-type: none"> If this parameter is set, in MySQL-compatible mode, the CREATE TABLE... LIKE syntax is empty.
mysql_compatibility_db=# SET behavior_compat_options = 'disable_including_all_mysql';
SET
mysql_compatibility_db=# CREATE TABLE mysql_copy LIKE mysql_like;
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
mysql_db=# \d+ mysql_copy;
Table "public.mysql_copy"
 <pre> Column Type Modifiers Storage Stats target Description -----+-----+-----+-----+-----+ id integer name character varying(10) extended score integer plain Has OIDs: no Distribute By: ROUND ROBIN Location Nodes: ALL DATANODES Options: orientation=row, compression=no </pre> | |
| cte_onetime_inline | <p>Indicates whether to execute inline for non-stream plans.</p> <ul style="list-style-type: none"> When this parameter is set, the CTE that is not in a stream plan and is referenced only once executes inline. If this parameter is not set, the CTE that is not in a stream plan and is referenced only once does not execute inline. | ORATDMySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------------|---|-------------------------------|
| skip_first_after_mysql | <p>Controls whether to ignore the FIRST/AFTER colname syntax in ALTER TABLE ADD/MODIFY/CHANGE COLUMN in MySQL-compatible mode.</p> <ul style="list-style-type: none">If this parameter is set, the FIRST/AFTER colname syntax is ignored, and executing this syntax will not result in any errors.
<pre>mysql_compatibility_db=# SET behavior_compat_options = 'skip_first_after_mysql';
mysql_compatibility_db=# ALTER TABLE t1 ADD COLUMN b text after a;
ALTER TABLE</pre>If this parameter is not set, the FIRST/AFTER colname syntax is not supported, and executing this syntax causes errors.
<pre>mysql_compatibility_db=# SET behavior_compat_options = '';
mysql_compatibility_db=# ALTER TABLE t1 ADD COLUMN b text after a;
ERROR: FIRST/AFTER is not yet supported.</pre> | MySQL |
| enable_division_by_zero_mysql | <p>Specifies whether division or modulo operations will result in an error when the divisor is 0 in MySQL-compatible mode. (This configuration item is supported only by clusters of version 8.1.3.110 or later.)</p> <ul style="list-style-type: none">If this parameter is set, NULL is returned if the divisor is 0 in a division or modulo operation.
<pre>compatible_mysql_db=# SET behavior_compat_options = 'enable_division_by_zero_mysql';
SET
compatible_mysql_db=# SELECT 1/0 AS test;
test

(1 row)</pre>If this parameter is not set, an error is returned if the divisor is 0 in a division or modulo operation.
<pre>compatible_mysql_db=# SELECT 1/0;
ERROR: division by zero</pre> | MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------------|--|-------------------------------|
| normal_session_id | <p>Indicates whether to generate a session ID in normal format.</p> <ul style="list-style-type: none">If this option is set, a session ID in normal format will be generated, which is compatible with session IDs in clusters of version 8.1.3 or earlier.
<pre>SET behavior_compat_options='normal_session_id';
SELECT pg_current_sessionid();
pg_current_sessionid

1660268184.140594655524608
(1 row)</pre>If this parameter is not set, a session ID in pretty format will be generated.
<pre>SET behavior_compat_options="";
SELECT pg_current_sessionid();
pg_current_sessionid

1660268184.140594655524608.coordinator1
(1 row)</pre> | ORA
TD
MySQL |
| disable_jsonb_exact_match | <p>Specifies whether to check the jsonb type during fuzzy match for binary operators.</p> <ul style="list-style-type: none">If this parameter is specified, operators search for matched items within the entire search scope (including the jsonb type) during fuzzy match. This setting is compatible with the match rules of cluster versions later than 8.1.1.
<pre>SET behavior_compat_options='disable_jsonb_exact_match';
select '2022' - '2'::text;
ERROR: cannot delete from scalar</pre>If this parameter is not specified, fuzzy match is performed within the search scope, except for the jsonb type. This setting is compatible with the match rules of clusters of version earlier than 8.1.1.
<pre>SET behavior_compat_options="";
select '2022' - '2'::text;
?column?

2020
(1 row)</pre> | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------------|--|-------------------------------|
| merge_into_with_trigger | <p>Controls whether the MERGE INTO operation can be performed on tables with triggers.</p> <ul style="list-style-type: none">• When this option is set, the MERGE INTO operation can be performed on tables with triggers. When the MERGE INTO operation is performed, the trigger on the table is not activated.• If this option is not set, an error is reported when the MERGE INTO operation is performed on a table with triggers. | ORA
TD
MySQL |
| add_column_default_v_function | <p>Controls whether expression in alter table add column default expression supports volatile functions.</p> <ul style="list-style-type: none">• If this option is selected, expression in alter table add column default expression supports volatile functions.• If this option is not selected, expression in alter table add column default expression does not support volatile functions. If expression contains volatile functions, an error will be reported during statement execution. | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-----------------------------|--|-------------------------------|
| disable_full_group_by_mysql | <p>Specifies whether to display non-aggregated function query columns after GROUP BY in a query.</p> <ul style="list-style-type: none">If this option is specified, the query does not display any non-aggregated function query columns after GROUP BY.
SET behavior_compat_options='disable_full_group_by_mysql';
SELECT a,b FROM t1 GROUP BY a;
a b
---+---
1 1
2 2
(2 rows)If this option is not specified, the query must display all non-aggregated function query columns after GROUP BY, or an error will be reported.
SET behavior_compat_options='';
SELECT a,b FROM t1 GROUP BY a;
ERROR: column "t1.b" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT a,b FROM t1 GROUP BY a; <p>CAUTION
This parameter must be used together with full_group_by_mode. For details, see full_group_by_mode.
After configuring this option, if full_group_by_mode is set to notpadding, non-aggregated query columns that are not part of the GROUP BY clause must have consistent data after grouping. Otherwise, the values in that column will be random.</p> | MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--|---|-------------------------------|
| disable_gc_fdw_filter_partial_pushdown | <p>Controls whether filter criteria are pushed down when querying data from a foreign table (of type gc_fdw) in a collaborative analysis scenario.</p> <ul style="list-style-type: none"> When this option is specified, if there are factors in the filter criteria that do not meet the pushdown conditions (such as non-immutable functions), all filter criteria will not be pushed down to ensure the consistency of the result set. This behavior is compatible with clusters of version earlier than 8.2.1. <pre>-- Create a table in the source cluster. CREATE TABLE t1(c1 INT, c2 INT, c3 INT) DISTRIBUTE BY HASH(c1); -- Create a foreign table with the same structure in the local cluster. CREATE SERVER server_remote FOREIGN DATA WRAPPER gc_fdw options(ADDRESS 'address', DBNAME 'dbname', USERNAME 'username', PASSWORD 'password'); CREATE FOREIGN TABLE t1(c1 INT, c2 INT, c3 INT) SERVER server_remote; -- Enable the parameter and see the pushdown behavior. SET behavior_compat_options = 'disable_gc_fdw_filter_partial_pushdown'; EXPLAIN (verbose on,costs off) SELECT * FROM t1 WHERE c1>3 AND c2 <100 AND now() - '20230101' < c3; QUERY PLAN ----- Streaming (type: GATHER) Output: c1, c2, c3 Node/s: All datanodes -> Foreign Scan on ca_schema.t1 Output: c1, c2, c3 Filter: ((t1.c1 > 3) AND (t1.c2 < 100) AND ((now() - '2023-01-01 00:00:00-08':timestamp with time zone) < (t1.c3)::interval)) Remote SQL: SELECT c1, c2, c3 FROM ca_schema.t1 (7 rows)</pre> <ul style="list-style-type: none"> If this parameter is not set, the pushable part of the filter criteria will be pushed down to the source cluster for execution, while the non-pushable part will be executed in the local cluster, which can improve the efficiency of foreign table queries. <pre>-- Disable this parameter and see the pushdown behavior. SET behavior_compat_options = ""; EXPLAIN (verbose on,costs off) SELECT * FROM t1 WHERE c1>3 AND c2 <100 AND now() - '20230101' < c3; QUERY PLAN ----- Streaming (type: GATHER)</pre> | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|------------------------------------|--|-------------------------------|
| | <pre> Output: c1, c2, c3 Node/s: All datanodes -> Foreign Scan on ca_schema.t1 Output: c1, c2, c3 Filter: ((now() - '2023-01-01 00:00:00-08':timestamp with time zone) < (t1.c3)::interval) Remote SQL: SELECT c1, c2, c3 FROM ca_schema.t1 WHERE ((c1 > 3)) AND ((c2 < 100)) (7 rows) </pre> | |
| ignore_unshipped_concurrent_update | <p>Determines whether to ignore new tuples when the UPDATE or DELETE statement is executed in the current session if the statement is not pushed down and the tuples are updated by other sessions. By default, new tuples are not processed.</p> <ul style="list-style-type: none"> If this parameter is specified, new tuples are ignored when the UPDATE or DELETE statement is executed in the current session. If the UPDATE or DELETE statement is successfully executed, data inconsistency occurs in concurrent update scenarios. This behavior is compatible with the behavior in versions earlier than 8.2.1. If this parameter is not set and the UPDATE or DELETE statement executed in the current session detects that tuples have been updated, the UPDATE or DELETE statement of the current session will be re-executed to ensure data consistency. The number of statement execution retries is controlled by the max_query_retry_times parameter. | ORA
TD
MySQL |
| disable_set_global_var_on_datanode | <p>Controls whether the set_config function can be used to set global variables on DNs.</p> <ul style="list-style-type: none"> When this parameter is set, the set_config function cannot be used to set global variables on DNs. By default, this behavior is compatible with the behavior in versions earlier than 8.2.1. If this parameter is not set, the set_config function can set global variables on DNs. As a result, the global variable values on CNs and DNs are inconsistent, and errors may occur when the read_global_var function is pushed down. | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------------------------|--|-------------------------------|
| variadic_null_check | <p>Whether variadic can transfer the NULL parameter. This function is disabled by default. This parameter is supported only by clusters of version 8.3.0 or later.</p> <ul style="list-style-type: none">When this parameter is set, passing NULL parameters to variadic is not allowed and will result in an error.
SET behavior_compat_options = 'variadic_null_check';

SELECT format ('array', VARIADIC NULL);
ERROR: VARIADIC parameter must be an array <p>NOTE
To be compatible with MySQL, enabling compat_concat_variadic does not take effect for the concat and concat_ws functions, and the NULL parameter can still be passed in.</p> <ul style="list-style-type: none">If this parameter is not set, NULL parameters can be passed to variadic.
SET behavior_compat_options = '';
SELECT format ('array', VARIADIC NULL);
format

array
(1 row) | ORATDMySQL |
| enable_use_syscol_in_replicate_table | <p>Specifies whether oid, ctid, tableoid, or xc_node_id can be used as filter, join, and having conditions during INSERT, UPDATE, MERGE INTO, and DELETE statements are executed on replication tables. This parameter is not set by default.</p> <ul style="list-style-type: none">If this parameter is not set and oid, ctid, tableoid, or xc_node_id is used as filter, join, or having conditions when the INSERT, UPDATE, MERGE INTO, or DELETE statements are executed on replication tables, the following error is reported:
ERROR: Can not use system column oid/ctid/tableoid/xc_node_id in Replication Table.When this parameter is set, the INSERT, UPDATE, MERGE INTO, and DELETE statements can be executed on replication tables using the system column id, ctid, tableoid, or xc_node_id. <p>CAUTION
If oid, ctid, tableoid, or xc_node_id is used as filter, join, and having conditions when the INSERT, UPDATE, MERGE INTO, or DELETE statements are executed on partition tables, the statement may result in cluster core dumps. In this case, exercise caution when setting this parameter.</p> | ORATDMySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------------------------|---|-------------------------------|
| enable_force_add_batch | <p>Determines whether DWS receives U packets in addbatch mode when support_batch_bind is set to on and enable_fast_query_shipping and enable_light_proxy are both set to off. This parameter is not set by default.</p> <ul style="list-style-type: none">• If this parameter is not set, support_batch_bind is set to on, and enable_fast_query_shipping and enable_light_proxy are both set to off, DWS does not receive U packets in addbatch mode.• If this parameter is set, support_batch_bind is set to on, and enable_fast_query_shipping and enable_light_proxy are both set to off, DWS receives U packets in addbatch mode. However, packets are imported to the database slowly, which may cause insufficient memory. So, exercise caution when setting this parameter. | ORA
TD
MySQL |
| disable_merge_sort_without_material | <p>Controls whether the current stream segment contains materialized operators. If it is, merge sort is used.</p> <ul style="list-style-type: none">• If this parameter is set and the current stream segment contains materialized operators (material, sort, agg, and CteScan), merge sort can be used. Otherwise, merge sort cannot be used.• If this parameter is unset, there is no need to verify whether the current stream segment contains materialized operators to determine whether to use merge sort. | ORA
TD
MySQL |
| enable_push_down_groupingset_subquery | <p>Specifies whether conditions from the outer query that are only related to a subquery can be pushed down to the subquery when the subquery contains a grouping set.</p> <ul style="list-style-type: none">• If the subquery contains grouping sets and this parameter is set, the conditions in the outer query cannot be pushed down to the subquery.• If the subquery contains grouping sets and this parameter is not set, the conditions in the outer query can be pushed down to the subquery. | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------|---|-------------------------------|
| enable_whole_row_var | <p>This parameter mainly involves two scenarios: 1. controlling whether tables or views are allowed to appear in SQL expressions, including but not limited to the target list of queries and GROUP BY lists. 2. controlling whether non-table records are allowed to appear in SQL expressions. This parameter is supported only by clusters of version 8.3.0 or later.</p> <ul style="list-style-type: none">When this parameter is set, tables or views are allowed to appear in SQL expressions.
SET behavior_compat_options = 'enable_whole_row_var';
SELECT a1 FROM t a1;
a1

(0 rows)
SELECT t FROM (SELECT 1) as t;
t

(1)
(1 rows)If this parameter is unset, tables or views are not allowed to appear in SQL expressions.
SET behavior_compat_options = "";
SELECT a1 FROM t a1;
ERROR: Table or view cannot appear in expression. Table/view name: t, alias: a1. Please check targetList, groupClause etc.
SELECT t FROM (SELECT 1) as t;
ERROR: Non-table records cannot appear in expression. alias: t. Please check targetList, groupClause etc. | ORA
TD
MYSQL |
| enable_unknown_datatype | <p>Specifies whether tables containing unknown columns can be created. This parameter is supported only by clusters of version 8.3.0 or later.</p> <ul style="list-style-type: none">When this parameter is set, tables containing unknown columns can be created.
SET behavior_compat_options = 'enable_unknown_datatype';
CREATE TABLE t(a unknown);
WARNING: column "a" has type "unknown"
DETAIL: Proceeding with relation creation anyway.
CREATE TABLEIf this parameter is unset, tables containing unknown columns cannot be created. If the table creation SQL contains an unknown column, an error will be reported.
SET behavior_compat_options = "";
create table t(a unknown);
ERROR: column "a" has type "unknown" | ORA
TD
MYSQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-----------------------------------|---|-------------------------------|
| alter_distribute_key_by_partition | <p>Specifies whether INSERT INTO is executed by partition when ALTER TABLE is used to modify the distribution column of a partitioned table.</p> <ul style="list-style-type: none">• If this parameter is set, INSERT INTO is executed by partition. The memory usage decreases but the performance deteriorates.• If this parameter is unset, INSERT INTO is performed on the entire partitioned table. The performance is good but the memory usage is high. | ORA
TD
MySQL |
| disable_update_returning_check | <p>Specifies whether to prevent multiple joins when a replication table is updated with the returning statement. This parameter is supported only by clusters of version 8.3.0 or later.</p> <ul style="list-style-type: none">• If the parameter is not set, the following error is reported when updating a replication table with a returning statement and involving multiple joins:
ERROR: Unsupported FOR UPDATE replicated table joined with other table.• Setting this parameter ensures backward compatibility with earlier versions. However, when updating a replication table with a returning statement and involving multiple joins, there may be inconsistencies in the result set. | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------------|--|-------------------------------|
| check_function_shippable | <p>Controls the check of the custom plpgsql/SQL function attributes. This parameter is supported only by clusters of version 8.3.0 or later.</p> <ul style="list-style-type: none">• If this parameter is not specified, the IMMUTABLE/STABLE/VOLATILE attributes of a user-defined function are not checked.• If this parameter is specified, the IMMUTABLE/STABLE/VOLATILE attributes of user-defined functions are checked based on the following principles:<ul style="list-style-type: none">– Whitelist: For the three functions in DBMS_OUTPUT, skip the check_function_shippable check.– If a user-defined function contains DML statements and the outer layer is IMMUTABLE or SHIPPABLE, the function is pushed down. As a result, an error is reported.– If the outer layer of a user-defined function is SHIPPABLE and the inner layer is IMMUTABLE, the function passes the check.– If the outer layer of a user-defined function is SHIPPABLE, the inner layer is SHIPPABLE and not IMMUTABLE, the function passes the check.– If the outer layer of a user-defined function is SHIPPABLE but the inner layer is none of the above, an error is reported. <p>For example, when this parameter is specified, an error is reported in the following scenarios:</p> <pre>CREATE OR replace function func_ship(a int) returns int LANGUAGE plpgsql NOT FENCED SHIPPABLE AS \$function\$ begin perform test_ship(); return a; EXCEPTION WHEN OTHERS THEN return a; end \$function\$; select func_ship(a) from tt3; ERROR: parent function is shippable but child is not immutable or shippable.</pre> | ORA
TD
MYS
QL |

| Configuration Item | Behavior | Applicable Compatibility Mode | | | | | | | | | | | | | | | | |
|------------------------|--|-------------------------------|----|---|---------|---|-----------------|---|----------------------|----|----|---|----------------------|---|----------------------|---|----------------------|-----------------|
| enable_full_string_agg | <p>Specifies how string_agg(a, delimiter) over (partition by b order by c) behaves in different situations, such as using full or incremental aggregation in the window. This parameter is supported only by clusters of version 8.3.0 or later.</p> <p>If this parameter is not set, incremental aggregation is used. If this parameter is set, full aggregation is used. By default, this parameter is not set.</p> <pre>CREATE TABLE string_agg_dn_col(c1 int, c2 text) WITH(orientation = column) distribute by hash(c1); INSERT INTO string_agg_dn_col values(1, 'test'); INSERT INTO string_agg_dn_col values(1, 'haidian'); INSERT INTO string_agg_dn_col values(1, 'nanjing'); SELECT t.c1 AS c1, string_agg(t.c2, ',') OVER(PARTITION BY t.c1 ORDER BY t.c2) AS c2 FROM string_agg_dn_col t ORDER BY c2;</pre> <table><tr><td>c1</td><td>c2</td></tr><tr><td>1</td><td>haidian</td></tr><tr><td>1</td><td>haidian,nanjing</td></tr><tr><td>1</td><td>haidian,nanjing,test</td></tr></table> <p>(3 rows)</p> <pre>SET behavior_compat_options='enable_full_string_agg'; SELECT t.c1 AS c1, string_agg(t.c2, ',') OVER(PARTITION BY t.c1 ORDER BY t.c2) AS c2 FROM string_agg_dn_col t ORDER BY c2;</pre> <table><tr><td>c1</td><td>c2</td></tr><tr><td>1</td><td>haidian,nanjing,test</td></tr><tr><td>1</td><td>haidian,nanjing,test</td></tr><tr><td>1</td><td>haidian,nanjing,test</td></tr></table> <p>(3 rows)</p> | c1 | c2 | 1 | haidian | 1 | haidian,nanjing | 1 | haidian,nanjing,test | c1 | c2 | 1 | haidian,nanjing,test | 1 | haidian,nanjing,test | 1 | haidian,nanjing,test | ORACLE
MySQL |
| c1 | c2 | | | | | | | | | | | | | | | | | |
| 1 | haidian | | | | | | | | | | | | | | | | | |
| 1 | haidian,nanjing | | | | | | | | | | | | | | | | | |
| 1 | haidian,nanjing,test | | | | | | | | | | | | | | | | | |
| c1 | c2 | | | | | | | | | | | | | | | | | |
| 1 | haidian,nanjing,test | | | | | | | | | | | | | | | | | |
| 1 | haidian,nanjing,test | | | | | | | | | | | | | | | | | |
| 1 | haidian,nanjing,test | | | | | | | | | | | | | | | | | |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------|--|-------------------------------|
| enable_banker_round | <p>Specifies how numeric types round their values, using the rounding or banker method. This parameter is supported only by clusters of version 8.3.0 or later.</p> <p>Behaviors controlled by parameters include:</p> <ul style="list-style-type: none"> Type conversion working when INSERT INTO and ::xxx specify a type, such as integer types (int1, int2, int4, int8), any precision types (decimal, numeric, number), or money types. Rounding and conversion functions for the numeric type: round(xxx.xx,s), cast('xxx.xx',numeric), or to_char(xxx.xx,'xxx'). Mathematical calculation of the numeric type. <p>NOTE
The banker's rounding rule is as follows: if the digit to be rounded is greater than 5, round up; if it is less than 5, round down; if it is exactly 5 (followed by zeros), round to the nearest even number; if it is exactly 5 (followed by non-zero digits), round up directly.</p> <ul style="list-style-type: none"> If this parameter is set, rounding uses the banker method. <pre>SET behavior_compat_options = enable_banker_round; SELECT 1.5::int1,1.5::int2,1.5::int4,1.5::int8,1.5::numeric(10,0),1.115::money; int1 int2 int4 int8 numeric money -----+-----+-----+-----+----- 2 2 2 2 2 \$1.12 SELECT 0.5::int1,0.5::int2,0.5::int4,0.5::int8,0.5::numeric(10,0),1.105::money; int1 int2 int4 int8 numeric money -----+-----+-----+-----+----- 0 0 0 0 0 \$1.10 SELECT round(1.05,1),round(1.15,1),cast('1.05',numeric(10,1)),cast('1.15',numeric(10,1)),to_char(1.05,'9D9'),to_char(1.15,'9D9'); round round numeric numeric to_char to_char -----+-----+-----+-----+-----+----- 1.0 1.2 1.0 1.2 1.0 1.2</pre> <ul style="list-style-type: none"> If this parameter is set, rounding uses round-off method. <pre>SET behavior_compat_options = ""; SELECT 1.5::int1,1.5::int2,1.5::int4,1.5::int8,1.5::numeric(10,0),1.115::money; int1 int2 int4 int8 numeric money -----+-----+-----+-----+----- 2 2 2 2 2 \$1.12 SELECT 0.5::int1,0.5::int2,0.5::int4,0.5::int8,0.5::numeric(10,0),1.105::money; int1 int2 int4 int8 numeric money -----+-----+-----+-----+----- 1 1 1 1 1 \$1.11 SELECT</pre> | ORA
TD
MYSQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------------------|--|-------------------------------|
| | <pre>round(1.05,1),round(1.15,1),cast('1.05',numeric(10,1)),cast('1.15',numeric(10,1)),to_char(1.05,'9D9'),to_char(1.15,'9D9');</pre> <pre>round round numeric numeric to_char to_char</pre> <pre>-----+-----+-----+-----+-----+-----</pre> <pre>1.1 1.2 1.1 1.2 1.1 1.2</pre> | |
| create_partition_local_index | <p>Controls whether the default index created on a partitioned table is a global index or a local index. This parameter is supported only by clusters of version 8.2.1.210 or later.</p> <ul style="list-style-type: none"> If this parameter is disabled, a global index is created on a partitioned table by default.
 <pre>SET behavior_compat_options = "";</pre> <pre>CREATE INDEX sale_id_idx ON sales(sale_id);</pre> <pre>ERROR: partitioned table does not support global index</pre> <pre>HINT: please set behavior_compat_options = 'create_partition_local_index'to create local index by default.</pre> If this parameter is enabled, a local index is created on a partitioned table by default.
 <pre>SET behavior_compat_options = create_partition_local_index;</pre> <pre>CREATE INDEX sale_id_idx ON sales(sale_id);</pre> <pre>CREATE INDEX</pre> | ORA |
| enable_integer_division_by_truncate | <p>Controls whether the integer division behavior result set returns integers or floating point numbers and the option is compatible with PG or ORA behaviors.</p> <ul style="list-style-type: none"> If this parameter is set, the integer division result is an integer, the decimal places are truncated, and this parameter is compatible with PG behaviors.
 <pre>SET behavior_compat_options = 'enable_integer_division_by_truncate';</pre> <pre>SELECT 8::int8 / 3::int8, 8::int4 / 3::int4, 8::int2 / 3::int2, 8::int1 / 3::int1;</pre> <pre>?column? ?column? ?column? ?column?</pre> <pre>-----+-----+-----+-----</pre> <pre>2 2 2 2</pre> <pre>(1 row)</pre> If this parameter is unset, the integer division result returns a floating point number, including decimal places, and this parameter is compatible with ORA behaviors.
 <pre>SET behavior_compat_options = "";</pre> <pre>SELECT 8::int8 / 3::int8, 8::int4 / 3::int4, 8::int2 / 3::int2, 8::int1 / 3::int1;</pre> <pre>?column? ?column? ?column? ?column?</pre> <pre>-----+-----+-----+-----</pre> <pre>2.666666666666667 2.666666666666667 2.666666666666667 </pre> <pre>2.666666666666667</pre> <pre>(1 row)</pre> | ORA
TD
MYS
QL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------------------|--|-------------------------------|
| select_into_allow_multi_result | <p>Controls whether SELECT INTO in a stored procedure can receive multiple rows of results or no result set at all.</p> <ul style="list-style-type: none">If this parameter is unset, the following error is reported when the SELECT INTO statement is used to insert multiple values or empty values in a stored procedure:
<pre>CREATE TABLE PersonTmpTable (id int primary key, name varchar(64), age int, city varchar(512) default null, update_time timestamp default null); INSERT INTO PersonTmpTable VALUES(1,'zhangsan', 23, 'wuhan', '2022-12-10 15:39:32'); INSERT INTO PersonTmpTable VALUE(2,'lisi', 11, 'beijing', '2022-12-13 15:39:32'); INSERT INTO PersonTmpTable VALUE(3,'wangwu', 46, 'xian', '2022-12-1 15:39:32'); INSERT INTO PersonTmpTable VALUE(4,'zhaoliu', 46, 'wuhan', '2022-12-31 15:39:32'); SET behavior_compat_options = ''; CREATE OR REPLACE function func_test1 RETURNS int LANGUAGE plpgsql AS \$\$DECLAREtmp INTEGER;BEGINSELECT id INTO tmp FROM PersonTmpTable;dbms_output.put_line(tmp);return tmp;END;\$ \$select func_test1(); ERROR: query returned no rows when process INTO. ERROR: query returned 2 rows more than one row.</pre>If this parameter is set, the first row or empty row is inserted when the SELECT INTO statement is used to insert multiple values or empty values in a stored procedure.
<pre>SET behavior_compat_options = 'select_into_allow_multi_result'; CREATE OR REPLACE function func_test1 RETURNS int LANGUAGE plpgsql AS \$\$DECLAREtmp INTEGER;BEGINSELECT id INTO tmp FROM PersonTmpTable;dbms_output.put_line(tmp);return tmp;END;\$ \$select func_test1(); CREATE FUNCTION</pre> | ORA
TD
MYS
QL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------------|---|-------------------------------|
| orderby_null_first | <p>Controls whether NULL values are treated as the minimum value by default when sorting with ORDER BY.</p> <ul style="list-style-type: none">If this parameter is set, NULL values are treated as the minimum value by default when sorting with ORDER BY.
SET behavior_compat_options = 'orderby_null_first';
SELECT * FROM test ORDER BY a;
a b
---+---
 1
1 2If this parameter is unset, NULL values are treated as the maximum value by default when sorting with ORDER BY.
SET behavior_compat_options = '';
SELECT * FROM test ORDER BY a;
a b
---+---
1 2
 1 | TD |
| unsupported_set_function_case | <p>Specifies whether multiple result set functions can be returned in a CASE WHEN condition. This is supported only by clusters of version 8.3.0.100 or later. This is enabled by default in newly installed clusters of version 9.1.0 or later.</p> <ul style="list-style-type: none">If this parameter is set, column storage does not support multiple result set functions in a CASE WHEN condition.
CREATE TABLE t1(id int, c1 text) with(orientation=column);
INSERT INTO t1 values(1, 'a#1');
SET behavior_compat_options = 'unsupported_set_function_case';
SELECT CASE split_part(regexp_split_to_table(c1, E'\,','#'),1) when 'a' then c1 else null end from t1;
ERROR: set-valued function called in context that cannot accept a setIf this parameter is not set, column storage supports multiple result set functions in a CASE WHEN condition.
SET behavior_compat_options = '';
SELECT CASE split_part(regexp_split_to_table(c1, E'\,','#'),1) when 'a' then c1 else null end from t1;
case

a#1
(1 row) | ORA
TD
MYSQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------------|--|-------------------------------|
| enable_change_search_path | <p>Specifies whether the search path can be modified after forming a general plan generic_plan. This is supported only by clusters of version 9.1.0 or later.</p> <ul style="list-style-type: none">When this parameter is not set, if a new search path is set and an EXECUTE statement is executed, the database will still search for the corresponding table under the original schema of the table. <pre>CREATE SCHEMA s1 CREATE TABLE abc(f1 INT); CREATE SCHEMA s2 CREATE TABLE abc(f1 INT); SET search_path = s1; INSERT INTO s1.abc VALUES(123);INSERT INTO s2.abc VALUES(456); SET search_path = s1; PREPARE p1 AS SELECT f1 FROM abc; EXECUTE p1; f1 ----- 123 (1 row) SET search_path = s2; SELECT f1 FROM abc; f1 ----- 456 (1 row) EXECUTE p1; f1 ----- 123 (1 row)</pre> <ul style="list-style-type: none">When this parameter is set, if a new search path is set and an EXECUTE statement is executed, the database will search for the corresponding table in the newly set search path. <pre>SET behavior_compat_options = 'enable_change_search_path'; EXECUTE p1; f1 ----- 456 (1 row) SET search_path = s1; EXECUTE p1; f1 ----- 123 (1 row)</pre> | TD |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-----------------------------|--|-------------------------------|
| enable_varchar_to_nvarchar2 | <p>Specifies whether varchar fields created or updated through DDL statements are automatically switched to nvarchar2 fields. This is supported only by clusters of version 9.1.0 or later.</p> <ul style="list-style-type: none">If this parameter is set, varchar fields created or updated through DDL statements are automatically switched to nvarchar2 fields.If this parameter is unset, varchar fields created or updated through DDL statements are not automatically switched to nvarchar2 fields. | ORA
TD
MySQL |
| normalize_negative_zero | <p>Specifies whether the ceil() and round() functions return -0 when processing specific values of the float type.</p> <ul style="list-style-type: none">When this parameter is set, the ceil() function returns 0 when processing (-1,0), and the round() function returns 0 when processing [-0.5, 0).
SET behavior_compat_options='normalize_negative_zero';
SELECT ceil(cast(-0.1 as float));
ceil

0
(1 row)
SELECT round(cast(-0.1 as FLOAT));
round

0
(1 row)When this parameter is not set, the ceil() function returns -0 when processing (-1,0), and the round() function returns -0 when processing [-0.5, 0).
SET behavior_compat_options = "";
SELECT ceil(cast(-0.1 as FLOAT));
ceil

-0
(1 row)
SELECT round(cast(-0.1 as FLOAT));
round

-0
(1 row) | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|---------------------------------|---|-------------------------------|
| disable_client_detection_commit | <p>Specifies whether to check there is a connection with the client before each transaction is committed. If the connection does not exist, an error is reported, the transaction is rolled back, and data duplication caused by repeated issuance due to disconnection is prevented.</p> <ul style="list-style-type: none">• If this parameter is not set, the system checks the existence of the client connection before each transaction is committed.• If this parameter is set, the system does not check the existence of the client connection before each transaction is committed. | ORA
TD
MySQL |
| change_illegal_char | <p>Specifies the display of illegal UTF8 characters when reading with GDS. This parameter is supported only by clusters of version 8.3.0.100 or later.</p> <p>When this parameter is enabled, illegal UTF8 characters that are incompatible with GDS are displayed as "◆" instead of "?".</p> | MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-------------------------|---|-------------------------------|
| row_use_pseudo_name | <p>Specifies whether row-related expressions generate pseudo column names for anonymous columns. This is supported only by 9.1.0.100 and later cluster versions.</p> <ul style="list-style-type: none">When this parameter is not set, if there is a corresponding real column name in the row expression, the real column name is used. If it is an anonymous column, pseudo column names f1, f2...fn are generated. <pre>SELECT row_to_json(row(1,'foo'));
row_to_json

{ "f1":1,"f2":"foo" }
(1 row)</pre> <pre>CREATE TABLE json_tbl(id INT, x INT, y text) WITH (ORIENTATION =
COLUMN);
INSERT INTO json_tbl VALUES (1, 1, 'txt1'), (2, 2, 'txt2'), (3, 3, 'txt3');
SELECT to_json(t.*) FROM json_tbl t;
to_json

{ "id":3,"x":3,"y":"txt3" }
{ "id":1,"x":1,"y":"txt1" }
{ "id":2,"x":2,"y":"txt2" }
(3 rows)</pre> <ul style="list-style-type: none">When this parameter is set, pseudo column names f1, f2...fn are generated for anonymous columns under the column storage condition in the row expression. <pre>SET behavior_compat_options='ROW_USE_PSEUDO_NAME';
SELECT to_json(t.*) FROM json_tbl t;
to_json

{ "f1":3,"f2":3,"f3":"txt3" }
{ "f1":1,"f2":1,"f3":"txt1" }
{ "f1":2,"f2":2,"f3":"txt2" }
(3 rows)</pre> | ORA
TD
MySQL |
| enable_trunc_orc_string | <p>Controls the foreign table query behavior when the foreign table field is in ORC format and the data type is varchar(n), but the field type in the ORC file is string and the length of the string exceeds n.</p> <ul style="list-style-type: none">If this parameter is not set, an error message is returned, indicating that the field is too long.If this parameter is set, the query is responded to, and the result is truncated by the length defined by varchar(n). | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--|---|-------------------------------|
| <code>gds_fill_multi_missing_fields</code> | <p>Controls the behavior when the GDS foreign table fault tolerance parameter fill_missing_fields is set to true or on. When fill_missing_fields is set to true or on in a GDS foreign table, any missing columns at the end of a row in the data source file are automatically set to NULL. Before this, only the last column in a row of the data source file can be missing without an error being reported.</p> <ul style="list-style-type: none">• If this option is specified, the GDS foreign table tolerates the missing of multiple last columns in a row of the source data file.• If this option is not specified, only the missing of the last column in a row of the data source file is tolerated in the GDS foreign table. This parameter compatible with historical behavior. | ORA
TD
MySQL |
| <code>correct_convert_tz</code> | <p>Controls whether the query result of the convert_tz function is consistent with the MySQL behavior. This is supported only by 9.1.0.210 and later cluster versions.</p> <ul style="list-style-type: none">• When this configuration item is enabled, the convert_tz function produces the expected results during time zone conversion, aligning with MySQL's behavior.
<pre>set behavior_compat_options = 'correct_convert_tz'; select convert_tz('2020-1-1 12:00', '+00:00', 'PRC'); convert_tz ----- 2020-01-01 20:00:00 (1 row)</pre>• If this configuration item is disabled, the results of the convert_tz function will be inconsistent with MySQL, failing to meet expectations.
<pre>set behavior_compat_options = ''; select convert_tz('2020-1-1 12:00', '+00:00', 'PRC'); convert_tz ----- 2020-01-01 04:00:00 (1 row)</pre> | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|--------------------------|--|-------------------------------|
| set_timezone_mysql_style | <p>Controls whether the specified timezone setting using the set timezone=±'Hour:Minute' command, aligns with MySQL's behavior. For example, set timezone='+08:00'/'-07:00'. This is supported only by 9.1.0.210 and later cluster versions.</p> <ul style="list-style-type: none">When configuration item is enabled, set timezone='+08:00' indicates that the time zone is set to GMT+8, aligning with MySQL's behavior.
<pre>set behavior_compat_options = 'set_timezone_mysql_style'; set timezone='UTC'; select * from t; a ----- 2020-01-01 12:00:00+00 (1 row) set timezone='+08:00'; select * from t; a ----- 2020-01-01 20:00:00+08 (1 row)</pre>If this configuration item is not enabled, set timezone='+08:00' indicates that the time zone is set to GMT-8, which aligns with PostgreSQL behavior, distinct from MySQL's behavior.
<pre>set behavior_compat_options = ''; set timezone='UTC'; select * from t; a ----- 2020-01-01 12:00:00+00 (1 row) set timezone='+08:00'; select * from t; a ----- 2020-01-01 04:00:00-08 (1 row)</pre> | ORAD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|----------------------------|---|-------------------------------|
| parquet_int96_is_localtime | <p>Specifies whether to use the database time zone to correct the original timestamp when the INT96 data (dedicated for timestamps) in a foreign table in Parquet format is read. It is available only for clusters of version 9.1.0.210 or later.</p> <ul style="list-style-type: none">If this configuration item is enabled, the parquet INT96 type stores the local time (not UTC+0), and the time zone is not corrected in the read results.
set behavior_compat_options = 'parquet_int96_is_localtime';
set time zone 'Asia/Shanghai';
select * from test_parquet_int96;
a

2025-01-25 00:00:00
(1 row)
set time zone 'Atlantic/Cape_Verde';
select * from test_parquet_int96;
a

2025-01-25 00:00:00
(1 row)If this configuration item is disabled, Parquet INT96 stores time in UTC+0. To ensure compatibility with engines like Spark and Hive, the database's time zone is used to correct the read results.
set behavior_compat_options = '';
set time zone 'Asia/Shanghai';
select * from test_parquet_int96;
a

2025-01-25 08:00:00
(1 row)
set time zone 'Atlantic/Cape_Verde';
select * from test_parquet_int96;
a

2025-01-24 23:00:00
(1 row) | ORA
TD
MySQL |

| Configuration Item | Behavior | Applicable Compatibility Mode |
|-----------------------------------|--|-------------------------------|
| enable_vacuum_update_datfrozenxid | <p>Specifies whether to update the datfrozenxid64 field during VACUUM FULL.</p> <p>This item is available in clusters of version 9.1.0.210 or later.</p> <p>Ensure that autovacuum is enabled. If it is disabled, the datfrozenxid64 field will be updated even if this item is not added. If there are many tables in the database, setting this item may slow down the VACUUM FULL operation on small tables.</p> <ul style="list-style-type: none">• If this item is set, VACUUM FULL updates the datfrozenxid64 field.• If this item is not set, VACUUM FULL does not update the datfrozenxid64 field by default. | ORA
TD
MySQL |

enable_matview

Parameter description: Specifies whether to enable the materialized view function.

Type: USERSET

Value range: Boolean

- **on:** The materialized view function is enabled.
- **off:** The materialized view function is disabled.

Default value: off

mv_max_cascade_depth

Parameter description: Specifies the maximum depth of nested materialized views. This parameter is available only for clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: an integer ranging from 1 to 10

Default value: 3

hive_compat_options

Parameter description: Specifies whether to convert an empty string to **NULL** when data is inserted into a numeric field using **INSERT INTO VALUES** or into a

character field using **INSERT INTO SELECT** in MySQL mode of DWS. This parameter is available only for clusters of version 9.1.0.210 or later.

For the examples of setting this parameter, see "INSERT" in *SQL Syntax Reference*.

Type: USERSET

Value range: enumerated values

- **empty_str_to_null:** An empty string will be converted to **NULL** when data is inserted into a numeric field or a character field.
- **empty:** When data is inserted into a numeric field using **INSERT INTO VALUES** or into a character field using **INSERT INTO SELECT**, empty strings are not processed.

Default value: empty

19.17 Fault Tolerance

This section describes parameters used for controlling the methods that the server processes an error occurring in the database system.

exit_on_error

Parameter description: Specifies whether to terminate the current session.

Type: SUSET

Value range: Boolean

- **on** indicates that any error will terminate the current session.
- **off** indicates that only a FATAL error will terminate the current session.

Default value: off

omit_encoding_error

Parameter description: When performing character encoding conversion in the database, if a character encoding error occurs and the target character set encoding is UTF-8, the converted character with the error can be ignored and replaced with "?".

Type: USERSET

Value range: Boolean

- **on** indicates that characters that have conversion errors will be ignored and replaced with question marks (?), and error information will be recorded in logs.
- **off** indicates that characters that have conversion errors cannot be converted and error information will be directly displayed.

Default value: off

max_query_retry_times

Parameter description: Specifies the maximum number of retries for the automatic retry feature when a SQL statement encounters an error. Currently, the supported error types for retry include **Connection reset by peer**, **Lock wait timeout**, and **Connection timed out**. Setting this parameter to **0** will disable the retry feature.

Type: USERSET

Value range: an integer ranging from 0 to 20

Default value: 6

retry_ecode_list

Parameter description: Specifies the list of SQL error types that support automatic retry.

Type: USERSET

Value range: a string

Default value: YY001 YY002 YY003 YY004 YY005 YY006 YY007 YY008 YY009
YY010 YY011 YY012 YY013 YY014 YY015 53200 08006 08000 57P01 XX003 XX009
YY016 CG003 CG004 F0011 F0012 45003 42P30

19.18 Connection Pool Parameters

When a connection pool is used to access the database, database connections are established and then stored in the memory as objects during system running. When you need to access the database, no new connection is established. Instead, an existing idle connection is selected from the connection pool. After you finish accessing the database, the database does not disable the connection but puts it back into the connection pool. The connection can be used for the next access request.

max_pool_size

Parameter description: Specifies the maximum number of connections between a CN's connection pool and another CN/DN.

Type: POSTMASTER

Value range: an integer ranging from 1 to 65535

Default value: **800** for CNs and **5000** for DN

persistent_datanode_connections

Parameter description: Specifies whether to release the connection for the current session.

Type: USERSET

Value range: Boolean

- **off** indicates that the connection for the current session will be released.
- **on** indicates that the connection for the current session will not be released.

NOTICE

After this function is enabled, a session may hold a connection but does not run a query. As a result, other query requests fail to be connected. To fix this problem, the number of sessions must be less than or equal to **max_active_statements**.

Default value: off

cache_connection

Parameter description: Specifies whether to reclaim the connections of a connection pool.

Type: USERSET

Value range: Boolean

- **on** indicates that the connections of a connection pool will be reclaimed.
- **off** indicates that the connections of a connection pool will not be reclaimed.

Default value: on

enable_force_reuse_connections

Parameter description: Specifies whether a session forcibly reuses a new connection.

Type: USERSET

Value range: Boolean

- **on** indicates that the new connection is forcibly used.
- **off** indicates that the current connection is used.

Default value: off

syscache_clean_policy

Parameter description: Specifies the policy for clearing the memory and number of idle DN connections. This is supported only by clusters of version 9.1.0.100 or later.

Type: SIGHUP

Value range: a string

This parameter policy consists of three values:

1. The first value ranges from 0 to 1 and represents the percentage of total available memory used by DNs. When the percentage of used memory reaches this value, 1/4 of the stream threads will be cleared, and the second value will be evaluated.

2. The second value ranges from 0 to 1 and represents the percentage of total available memory used by syscache on DNs. When the percentage of syscache memory usage reaches this value, the third value will be evaluated.
3. The third value ranges from 0 to INT_MAX and is measured in MB. It represents the size of syscache memory used by idle threads. When the syscache memory usage of an idle thread reaches this value, the syscache used by that thread will be cleared.

Default value: 0.8,0.3,64

NOTICE

- Before setting this parameter, evaluate the memory usage using views [PV_SESSION_MEMORY_DETAIL](#) and [PV_TOTAL_MEMORY_DETAIL](#).
 - When setting this parameter, follow the specified format, ensuring that the three values are separated by commas without spaces.
 - If the parameter is not set according to the specified format and the setting fails, a WARNING log will be generated in the log, and the parameter value displayed when using the SHOW command to query the parameter will be the last successfully set value. If the setting fails and the system is restarted, the parameter will be set to the default value.
 - During the Readcommand phase, if a thread on CN times out after 30 seconds, it will clear DNs if syscache is greater than 256 MB. There are two operations:
 1. If the overall memory usage reaches 80%, an auxiliary thread will monitor the memory usage and clear 1/4 of the stream threads. It will also check if syscache usage exceeds 30% of the total memory usage. If it does, it will clear the syscache of Readcommand phase pg threads greater than 64 MB.
 2. If a stream thread is idle for more than 30 seconds and syscache usage is greater than 64 MB, it will clear the syscache.
-

19.19 Cluster Transaction Parameters

This section describes the settings and value ranges of cluster transaction parameters.

transaction_isolation

Parameter description: Specifies the isolation level of the current transaction.

Type: USERSET

Value range:

- **READ COMMITTED:** Only committed data is read. This is the default.
- **READ UNCOMMITTED:** DWS does not support **READ UNCOMMITTED**. If **READ UNCOMMITTED** is set, **READ COMMITTED** is executed instead.
- **REPEATABLE READ:** Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.

- **SERIALIZABLE**: DWS does not support **SERIALIZABLE**. If **SERIALIZABLE** is set, **REPEATABLE READ** is executed instead.

Default value: READ COMMITTED

transaction_read_only

Parameter description: Specifies that the current transaction is a read-only transaction.

Type: USERSET

Value range: Boolean

- **on** indicates that the current transaction is a read-only transaction.
- **off** indicates that the current transaction can be a read/write transaction.

Default value: off for CNs and on for DN

xc_maintenance_mode

Parameter description: Specifies whether the system is in maintenance mode.

Type: SUSERSET

Value range: Boolean

- **on** indicates that maintenance mode is enabled.
- **off** indicates that the maintenance mode is disabled.

Default value: off

NOTICE

Enable the maintenance mode with caution to avoid cluster data inconsistencies.

allow_concurrent_tuple_update

Parameter description: Specifies whether to allow concurrent update.

Type: USERSET

Value range: Boolean

- **on** indicates it is enabled.
- **off** indicates it is disabled.

Default value: on

gtm_backup_barrier

Parameter description: Specifies whether to create a restoration point for the GTM starting point.

Type: SUSERSET

Value range: Boolean

- **on** indicates that a restoration point will be created for the GTM starting point.
- **off** indicates that a restoration point will not be created for the GTM starting point.

Default value: off

transaction_deferrable

Parameter description: Specifies whether to delay the execution of a read-only serial transaction without incurring an execution failure. Assume this parameter is set to **on**. When the server detects that the tuples read by a read-only transaction are being modified by other transactions, it delays the execution of the read-only transaction until the other transactions finish modifying the tuples. Currently, this parameter is not used in DWS. Similar to this parameter, the [default_transaction_deferrable](#) parameter is used to specify whether to allow delayed execution of a transaction.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution of a read-only serial transaction can be delayed.
- **off** indicates that the execution of a read-only serial transaction cannot be delayed.

Default value: off

enforce_two_phase_commit

Parameter description: This parameter is reserved for compatibility with earlier versions. This parameter is invalid in the current version.

enable_show_any_tuples

Parameter description: This parameter is available only in a read-only transaction and is used for analysis. When this parameter is set to **on/true**, all versions of tuples in the table are displayed.

Type: USERSET

Value range: Boolean

- **on/true** indicates that all versions of tuples in the table are displayed.
- **off/false** indicates that no versions of tuples in the table are displayed.

Default value: off

idle_in_transaction_timeout

Parameter description: duration during which a transaction is allowed to be in the idle state. When a transaction is in the idle state for a period specified by this parameter, the transaction is terminated. This function takes effect only for client

connections that are directly connected to CNs and does not take effect for direct DN connections or internal connections. This parameter is supported only by clusters of version 8.2.1.100 or later.

Type: USERSET

Value range: 0 to 86400, in second.

Default value: 3600

19.20 Developer Operations

enable_light_colupdate

Parameter description: Specifies whether to enable the lightweight column-store update.

Type: USERSET

Value range: Boolean

- **on** indicates that the lightweight column-store update is enabled.
- **off** indicates that the lightweight column-store update is disabled.

Default value: off

NOTE

There is a low probability that an error is reported when lightweight **UPDATE** and backend column-store **AUTOVACUUM** coexist. You can run **ALTER TABLE** to set the table-level parameter **enable_column_autovacuum_garbage** to **off** to avoid this issue. If the table-level parameter **enable_column_autovacuum_garbage** is set to **off**, the backend column-store **AUTOVACUUM** of the table is disabled.

enable_fast_query_shipping

Parameter description: Specifies whether to use the distributed framework for a query planner.

Type: USERSET

Value range: Boolean

- **on** indicates that execution plans are generated on CNs and DN connections separately.
- **off** indicates that the distributed framework is used. Execution plans are generated on CNs and then sent to DN connections for execution.

Default value: on

enable_trigger_shipping

Parameter description: Specifies whether the trigger can be pushed to DN connections for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the trigger can be pushed to DN for execution.
- **off** indicates that the trigger cannot be pushed to DN. It must be executed on the CN.

Default value: on

enable_remotejoin

Parameter description: Specifies whether JOIN operation plans can be delivered to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that JOIN operation plans can be delivered to DN for execution.
- **off** indicates that JOIN operation plans cannot be delivered to DN for execution.

Default value: on

enable_remotegroup

Parameter description: Specifies whether the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plans of **GROUP BY** and **AGGREGATE** can be delivered to DN for execution.
- **off** indicates that the execution plans of **GROUP BY** and **AGGREGATE** cannot be delivered to DN for execution.

Default value: on

enable_remotelimit

Parameter description: Specifies whether the execution plan specified in the LIMIT clause can be pushed down to DN for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plan specified in the LIMIT clause can be pushed down to DN for execution.
- **off** indicates that the execution plan specified in the LIMIT clause cannot be delivered to DN for execution.

Default value: on

enable_limit_stop

Parameter description: whether to enable the **early stop** optimization for **LIMIT** statements. For a **LIMIT n** statement, if **early stop** is enabled, the CN requests the

DN to end the execution after receiving *n* pieces of data. This method is applicable to complex queries with **LIMIT**. This parameter is supported only by clusters of version 8.1.3.320 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that **early stop** is enabled for LIMIT statements.
- **off** indicates that **early stop** is disabled for LIMIT statements.

Default value: on

enable_remotesort

Parameter description: Specifies whether the execution plan of the ORDER BY clause can be delivered to DNs for execution.

Type: USERSET

Value range: Boolean

- **on** indicates that the execution plan of the ORDER BY clause can be delivered to DNs for execution.
- **off** indicates that the execution plan of the ORDER BY clause cannot be delivered to DNs for execution.

Default value: on

enable_join_pseudoconst

Parameter description: Specifies whether joining with the pseudo constant is allowed. A pseudo constant indicates that the variables on both sides of a join are identical to the same constant.

Type: USERSET

Value range: Boolean

- **on** indicates that joining with the pseudo constant is allowed.
- **off** indicates that joining with the pseudo constant is not allowed.

Default value: off

cost_model_version

Parameter description: Specifies the model used for cost estimation in the application scenario. This parameter affects the distinct estimation of the expression, HashJoin cost model, estimation of the number of rows, distribution key selection during redistribution, and estimation of the number of aggregate rows.

Type: USERSET

Value range: 0, 1, 2, 3, or 4

- **0** indicates that the original cost estimation model is used.

- **1** indicates that the enhanced distinct estimation of the expression, HashJoin cost estimation model, estimation of the number of rows, distribution key selection during redistribution, and estimation of the number of aggregate rows are used on the basis of **0**.
- **2** indicates that the ANALYZE sampling algorithm with better randomness is used on the basis of **1** to improve the accuracy of statistics collection.
- **3** indicates that the broadcast cost estimation in large cluster scenarios is optimized based on **2** so that the optimizer can select a better plan. This option is supported only by clusters of version 8.3.0 or later.
- **4** indicates that in addition to the optimizations made to the cost estimation of hashjoin parallelization, skew, and column-store index ordering in **3**, there are also optimized row estimations for coalesce expressions and improved recognition of skew optimization for subquery constant output columns during joins.

Default value: 4

debug_assertions

Parameter description: Specifies whether to enable various assertion checks. This parameter assists in debugging. If you are experiencing strange problems or crashes, set this parameter to **on** to identify programming defects. To use this parameter, the macro USE_ASSERT_CHECKING must be defined (through the configure option **--enable-cassert**) during the DWS compilation.

Type: USERSET

Value range: Boolean

- **on** indicates that various assertion checks are enabled.
- **off** indicates that various assertion checks are disabled.

NOTE

This parameter is set to **on** by default if DWS is compiled with various assertion checks enabled.

Default value: off

distribute_test_param

Parameter description: Specifies whether the embedded test stubs for testing the distribution framework take effect. In most cases, developers embed some test stubs in the code during fault injection tests. Each test stub is identified by a unique name. The value of this parameter is a triplet that includes three values: thread level, test stub name, and error level of the injected fault. The three values are separated by commas (,).

Type: USERSET

Value range: a string indicating the name of any embedded test stub.

Default value: -1, default, default

ignore_checksum_failure

Parameter description: Sets whether to ignore check failures (but still generates an alarm) and continues reading data. This parameter is valid only if **enable_crc_check** is set to **on**. Continuing reading data may result in breakdown, damaged data being transferred or hidden, failure of data recovery from remote nodes, or other serious problems. You are not advised to modify the settings.

Type: SUSET

Value range: Boolean

- **on** indicates that data check errors are ignored.
- **off** indicates that data check errors are reported.

Default value: off

default_table_behavior

Parameter description: behavior type of the default table. This parameter is supported only by clusters of version 8.2.1 or later.

Type: USERSET

Value range: **column_btree_index**, **column_high_compress**, **column_middle_compress**, or **column_low_compress**

- **column_btree_index** indicates that the default index for creating a column-store table is **B-Tree**.
- **column_high_compress** indicates that the default compression level of column-store tables is **high**.
- **column_middle_compress** indicates that the default compression level of column-store tables is **middle**.
- **column_low_compress** indicates that the default compression level of column-store tables is **low**.

Default value: an empty string

enable_colstore

Parameter description: Specifies whether to create a table as a column-store table by default when no storage method is specified. The value for each node must be the same. This parameter is used for tests. Users are not allowed to enable it.

Type: SUSET

Value range: Boolean

Default value: off

enable_force_vector_engine

Parameter description: Specifies whether to forcibly generate vectorized execution plans for a vectorized execution operator if the operator's child node is a non-vectorized operator. When this parameter is set to **on**, vectorized execution

plans are forcibly generated. When **enable_force_vector_engine** is enabled, no matter it is a row-store table, column-store table, or hybrid row-column store table, if the plantree does not contain scenarios that do not support vectorization, the vectorized executor is forcibly used.

Type: USERSET

Value range: Boolean

Default value: off

enable_csqual_pushdown

Parameter description: Specifies whether to deliver filter criteria for a rough check during query.

Type: USERSET

Value range: Boolean

- **on** indicates that a rough check is performed with filter criteria delivered during query.
- **off** indicates that a rough check is performed without filter criteria delivered during query.

Default value: on

explain_dna_file

Parameter description: Specifies the name of a CSV file exported when **explain_perf_mode** is set to **run**.

Type: USERSET

NOTICE

The value of this parameter must be an absolute path plus a file name with the extension **.csv**.

Value range: a string

Default value: NULL

explain_perf_mode

Parameter description: Specifies the display format of the **explain** command.

Type: USERSET

Value range: **normal**, **pretty**, **summary**, and **run**

- **normal** indicates that the default printing format is used.
- **pretty** indicates that the optimized display mode of DWS is used. A new format contains a plan node ID, directly and effectively analyzing performance.

- **summary** indicates that the analysis result based on such information is printed in addition to the printed information in the format specified by **pretty**.
- **run** indicates that in addition to the printed information specified by **summary**, the database exports the information as a CSV file.

Default value: pretty

join_num_distinct

Parameter description: Controls the default distinct value of the join column or expression in application scenarios.

Type: USERSET

Value range: a double-precision floating point number greater than or equal to -100. Decimals may be truncated when displayed on clients.

- If the value is greater than 0, the value is used as the default distinct value.
- If the value is greater than or equal to -100 and less than 0, it means the percentage used to estimate the default distinct value.
- If the value is 0, the default distinct value is 200.

Default value: -20

outer_join_max_rows_multipler

Parameter description: Specifies the maximum number of estimated rows for outer joins.

Type: USERSET

Value range: 0 or a double-precision floating point number greater than or equal to 1. Decimals may be truncated when displayed on clients.

- If the value is 0, the estimated number of rows for outer joins is not limited.
- If the value is greater than or equal to 1, the estimated number of rows cannot exceed a multiple of the number of rows in the foreign table in the outer join.

Default value: 1.1

qual_num_distinct

Parameter description: Controls the default distinct value of the filter column or expression in application scenarios.

Type: USERSET

Value range: a double-precision floating point number greater than or equal to -100. Decimals may be truncated when displayed on clients.

- If the value is greater than 0, the value is used as the default distinct value.
- If the value is greater than or equal to -100 and less than 0, it means the percentage used to estimate the default distinct value.

- If the value is **0**, the default distinct value is **200**.

Default value: 200

trace_notify

Parameter description: Specifies whether to generate a large amount of debugging output for the **LISTEN** and **NOTIFY** commands. [client_min_messages](#) or [log_min_messages](#) must be **DEBUG1** or lower so that such output can be recorded in the logs on the client or server separately.

Type: USERSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: off

trace_sort

Parameter description: Specifies whether to display information about resource usage during sorting operations in logs. This parameter is available only when the macro **TRACE_SORT** is defined during the DWS compilation. However, **TRACE_SORT** is currently defined by default.

Type: USERSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

Default value: off

zero_damaged_pages

Parameter description: Specifies whether to detect a damaged page header that causes DWS to report an error, aborting the current transaction.

Type: SUSET

Value range: Boolean

- **on** indicates that the function is enabled.
- **off** indicates that the function is disabled.

 NOTE

- Setting this parameter to **on** causes the system to report a warning, pad the damaged page with zeros, and then continue with subsequent processing. This behavior will damage data, that is, all rows on the damaged page. However, it allows you to bypass the error and retrieve rows from any undamaged pages that are present in the table. Therefore, it is useful for restoring data that is damaged due to a hardware or software error. In most cases, you are not advised to set this parameter to **on** unless you do not want to restore data from the damaged pages of a table.
- For a column-store table, the system will skip the entire CU and then continue processing. The supported scenarios include the CRC check failure, magic check failure, and incorrect CU length.

Default value: off

replication_test

Parameter description: Specifies whether to enable internal testing on the data replication function.

Type: USERSET

Value range: Boolean

- **on** indicates that internal testing on the data replication function is enabled.
- **off** indicates that internal testing on the data replication function is disabled.

Default value: off

cost_param

Parameter description: Controls use of different estimation methods in specific customer scenarios, allowing estimated values approximating to onsite values. This parameter can control various methods simultaneously by performing AND (&) operations on the bit for each method. A method is selected if its value is not 0.

If **cost_param & 1** is not set to 0, an improvement mechanism is selected for calculating a non-equi join selection rate, which is more accurate in estimation of self-join (join between two same tables). In V300R002C00 and later, **cost_param & 1=0** is not used. That is, an optimized formula is selected for calculation.

When **cost_param & 2** is set to a value other than 0, the selection rate is estimated based on multiple filter criteria. The lowest selection rate among all filter criteria, but not the product of the selection rates for two tables under a specific filter criterion, is used as the total selection rate. This method is more accurate when a close correlation exists between the columns to be filtered.

When **cost_param & 4** is not 0, the selected debugging model is not recommended when the stream node is evaluated.

When **cost_param & 16** is not 0, the model between fully correlated and fully uncorrelated models is used to calculate the comprehensive selection rate of two or more filtering conditions or join conditions. If there are many filtering conditions, the strongly-correlated model is preferred.

Type: USERSET

Value range: an integer ranging from 1 to INT_MAX

Default value: 16

convert_string_to_digit

Parameter description: Specifies the implicit conversion priority, which determines whether to preferentially convert strings into numbers.

Type: USERSET

Value range: Boolean

- **on** indicates that strings are preferentially converted into numbers.
- **off** indicates that strings are not preferentially converted into numbers.

Default value: on

NOTICE

Modify this parameter only when absolutely necessary because the modification will change the rule for converting internal data types and may cause unexpected results.

nls_timestamp_format

Parameter description: Specifies the default timestamp format.

Type: USERSET

Value range: a string

Default value: DD-Mon-YYYY HH:MI:SS.FF AM

enable_partitionwise

Parameter description: Specifies whether to select an intelligent algorithm for joining partitioned tables.

Type: USERSET

Value range: Boolean

- **on** indicates that an intelligent algorithm is selected.
- **off** indicates that an intelligent algorithm is not selected.

Default value: off

enable_partition_dynamic_pruning

Parameter description: Specifies whether dynamic pruning is enabled during partition table scanning.

Type: USERSET

Value range: Boolean

- **on**: enable
- **off**: disable

Default value: on

max_user_defined_exception

Parameter description: Specifies the maximum number of exceptions. The default value cannot be changed.

Type: USERSET

Value range: an integer

Default value: 1000

datanode_strong_sync

Parameter description: This parameter no longer takes effect.

Type: USERSET

Value range: Boolean

- **on** indicates that forcible synchronization between stream nodes is enabled.
- **off** indicates that forcible synchronization between stream nodes is disabled.

Default value: off

enable_global_stats

Parameter description: Specifies the current statistics mode. This parameter is used to compare global statistics generation plans and the statistics generation plans for a single DN. This parameter is used for tests. Users are not allowed to enable it.

Type: SUSET

Value range: Boolean

- **on** or **true** indicates the global statistics mode.
- **off** or **false** indicates the single-DN statistics mode.

Default value: on

enable_fast_numeric

Parameter description: Specifies whether to enable optimization for numeric data calculation. Calculation of numeric data is time-consuming. Numeric data is converted into int64- or int128-type data to improve numeric data calculation performance.

Type: USERSET

Value range: Boolean

- **on/true** indicates that optimization for numeric data calculation is enabled.

- **off/false** indicates that optimization for numeric data calculation is disabled.

Default value: on

enable_row_fast_numeric

Parameter description: Specifies the format in which numeric data in a row-store table is spilled to disks.

Type: USERSET

Value range: Boolean

- **on/true** indicates that numeric data in a row-store table is spilled to disks in bigint format.
- **off/false** indicates that numeric data in a row-store table is spilled to disks in the original format.

NOTICE

If this parameter is set to **on**, you are advised to enable **enable_force_vector_engine** to improve the query performance of large data sets. However, compared with the original format, there is a high probability that the bigint format occupies more disk space. For example, the TPC-H test set occupies about 7% more space (reference value, may vary depending on the environment).

Default value: off

rewrite_rule

Parameter description: Specifies the rewriting rule for enabled optional queries. Some query rewriting rules are optional. Enabling them cannot always improve query efficiency. In a specific customer scenario, you can set the query rewriting rules through the GUC parameter to achieve optimal query efficiency.

This parameter can control the combination of query rewriting rules, for example, there are multiple rewriting rules: rule1, rule2, rule3, and rule4. To set the parameters, you can perform the following operations:

```
set rewrite_rule=rule1;      --Enable query rewriting rule rule1.
set rewrite_rule=rule2,rule3; --Enable query rewriting rules rule2 and rule3.
set rewrite_rule=none;       --Disable all optional query rewriting rules.
```

Type: USERSET

Value range: a string

- **none:** No optional query rewrite rules are used.
- **Lazyagg:** The Lazy Agg query rewrite rule is used to eliminate aggregate operations in subqueries.
- **magicset:** The Magic Set query rewrite rule is used to push conditions from the main query down to promoted sublinks.
- **uniquecheck:** Uses the Unique Check rewriting rule. (The scenario where the target column does not contain the expression sublink of the aggregate

function can be improved. The function can be enabled only when the value of the target column is unique after the sublink is aggregated based on the associated column. This function is recommended to be used by optimization engineers.)

- **disablerep**: Uses the function that prohibits pulling up sublinks of the replication table. (Disables sublink pull-up for the replication table.)
- **projection_pushdown**: the Projection Pushdown rewriting rule (Removes columns that are not used by the parent query from the subquery).
- **or_conversion**: the OR conversion rewriting rule (eliminates the association OR conditions that are inefficient to execute).
- **plain_lazyagg**: the **Plain Lazy Agg** query rewriting rule (eliminates aggregation operations in a single subquery). This option is supported only by clusters of version 8.1.3.100 or later.
- **eager_magicset**: Uses the **eager_magicset** query rewriting rule (to push down conditions from the main query to subqueries). This option is supported only by clusters of version 8.2.0 or later.
- **casewhen_simplification**: This rewrite rule uses the CASE WHEN statement to simplify queries. When enabled, it rewrites **(case when xxx then const1 else const2)=const1**. It simplifies the CASE WHEN statement and reduces redundant calculation to improve query performance. This option is supported only by clusters of version 8.3.0 or later.
- **outer_join_quality_imply**: When there is an equi-join condition between a left outer join and a right outer join, this rule pushes the expression condition on the outer table's join column down to the inner table's join column. This option is supported only by clusters of version 8.3.0 or later.
- **inlist_merge**: This query rewrite rule uses the **inlist_or_inlist** method to merge **OR** statements with the same base table column. When enabled, it merges and rewrites **(where a in (list1) or a in (list2))** to support **inlist2join**. This option is supported only by clusters of version 8.3.0 or later.
- **subquery_qual_pull_up**: For subqueries that cannot be promoted, if the subquery has filtering conditions on columns that are also used for joining with other tables, this rule extracts the filtering conditions from the subquery and passes them to the other side of the join condition. Currently, only **var op const** forms without type conversion, such as **a > 2**, are supported. When enabled, it is assumed that **outer_join_quality_imply** is also enabled. This is supported only by clusters of version 9.1.0 or later.
- **not_distinct_from_opt**: When this function is enabled, the **is not distinct from** expression is temporarily rewritten to an equal sign expression so that statements containing this syntax can generate HASH JOIN or MERGE JOIN plans, improving execution performance. This is supported only by clusters of version 9.1.0.218 or later.

Default value: **magicset**, **or_conversion**, **projection_pushdown**, **plain_lazyagg**, or **subquery_qual_pull_up**

mv_rewrite_rule

Parameter description: whether to enable the rewriting rule for the materialized view.

Type: USERSET

Value range: a string

- **none:** No materialized view rewriting rule is used. This value is available only in clusters of version 8.2.1.100 or later.
- **text:** materialized view rewriting rule that uses text matching. This value is available only in clusters of version 8.2.1.100 or later.
- **general:** structure matching. This value is supported only by 9.1.0.200 and later cluster versions.
- **predicate:** expression framework for structure matching rewriting. This parameter is used for preprocessing and regularization of filter and association conditions. It takes effect only when **general** is enabled. This value is supported only by 9.1.0.210 and later cluster versions.
- **view_delta:** used to use primary and foreign key relationships to eliminate redundant joins in materialized views and then match the materialized views with queries. This value is supported only by 9.1.0.210 and later cluster versions.
- **single:** The single-table materialized view uses the materialized view rewriting rule. This value is supported only by clusters of version 9.1.0.222 or later.

Default value: **text**, **general**, **predicate**, and **single**.

NOTICE

general, **predicate**, and **view_delta** are restricted for commercial use. To use them, contact technical support.

enable_compress_spill

Parameter description: Specifies whether to enable the compression function of writing data to a disk.

Type: USERSET

Value range: Boolean

- **on/true** indicates that optimization for writing data to a disk is enabled.
- **off/false** indicates that optimization for writing data to a disk is disabled.

Default value: **on**

analysis_options

Parameter description: Specifies whether to enable corresponding features, such as data validation and performance statistics.

Type: USERSET

Value range: a string

- **LLVM_COMPILE** indicates that the codegen compilation time of each thread is displayed on the explain performance page.

- **HASH_CONFLICT** indicates that the log file in the **pg_log** directory of the DN process displays the hash table statistics, including the hash table size, hash chain length, and hash conflict information.
- **STREAM_DATA_CHECK** indicates that a CRC check is performed on data before and after network data transmission.
- **TURBO_DATA_CHECK** indicates that the data context of the **ScalarVector** and **VectorBatch** operators of Turbo is verified. This parameter is supported only by clusters of version 8.3.0.100 or later.
- **KEEP_SAMPLE_DATA**: This parameter retains the sampling data used in each analyze operation in the form of temporary tables. This parameter is supported only by clusters of version 9.1.0 or later.
- **BLOCK_RULE**: indicates that the time required for checking the query filter is displayed on the explain performance page. This is supported only by 9.1.0.100 and later cluster versions.

Default value: **off(ALL)**, which indicates that no location function is enabled.

resource_track_log

Parameter description: Specifies the log level of self-diagnosis. Currently, this parameter takes effect only in multi-column statistics.

Type: USERSET

Value range: a string

- **summary**: Brief diagnosis information is displayed.
- **detail**: Detailed diagnosis information is displayed.

Currently, the two parameter values differ only when there is an alarm about multi-column statistics not collected. If the parameter is set to **summary**, such an alarm will not be displayed. If it is set to **detail**, such an alarm will be displayed.

Default value: **summary**

hll_default_log2m

Parameter description: Specifies the number of buckets for HLL data. The number of buckets affects the precision of distinct values calculated by HLL. As the number of buckets increases, the deviation becomes smaller. The deviation range is as follows: $[-1.04/2^{\log 2m^{1/2}}, +1.04/2^{\log 2m^{1/2}}]$

Type: USERSET

Value range: an integer ranging from 10 to 16

Default value: 11

hll_default_regwidth

Parameter description: Specifies the number of bits in each bucket for HLL data. A larger value indicates more memory occupied by HLL. **hll_default_regwidth** and **hll_default_log2m** determine the maximum number of distinct values that can be calculated by HLL. For details, see [Table 19-3](#).

Type: USERSET

Value range: an integer ranging from 1 to 5

Default value: 5

Table 19-3 Maximum number of calculated distinct values determined by hll_default_log2m and hll_default_regwidth

| log2m | regwidth = 1 | regwidth = 2 | regwidth = 3 | regwidth = 4 | regwidth = 5 |
|-------|--------------|--------------|--------------|--------------|--------------|
| 10 | 7.4e+02 | 3.0e+03 | 4.7e+04 | 1.2e+07 | 7.9e+11 |
| 11 | 1.5e+03 | 5.9e+03 | 9.5e+04 | 2.4e+07 | 1.6e+12 |
| 12 | 3.0e+03 | 1.2e+04 | 1.9e+05 | 4.8e+07 | 3.2e+12 |
| 13 | 5.9e+03 | 2.4e+04 | 3.8e+05 | 9.7e+07 | 6.3e+12 |
| 14 | 1.2e+04 | 4.7e+04 | 7.6e+05 | 1.9e+08 | 1.3e+13 |
| 15 | 2.4e+04 | 9.5e+04 | 1.5e+06 | 3.9e+08 | 2.5e+13 |

hll_default_expthresh

Parameter description: Specifies the default threshold for switching from the **explicit** mode to the **sparse** mode.

Type: USERSET

Value range: an integer ranging from -1 to 7 -1 indicates the auto mode; 0 indicates that the **explicit** mode is skipped; a value from 1 to 7 indicates that the mode is switched when the number of distinct values reaches $2^{\text{hll_default_expthresh}}$.

Default value: -1

hll_default_sparseon

Parameter description: Specifies whether to enable the **sparse** mode by default.

Type: USERSET

Valid value: 0 and 1 0 indicates that the **sparse** mode is disabled by default. 1 indicates that the **sparse** mode is enabled by default.

Default value: 1

hll_max_sparse

Parameter description: Specifies the size of **max_sparse**.

Type: USERSET

Value range: an integer ranging from -1 to INT_MAX

Default value: -1

enable_compress_hll

Parameter description: Specifies whether to enable memory optimization for HLL.

Type: USERSET

Value range: Boolean

- **on** or **true** indicates that memory optimization is enabled.
- **off** or **false** indicates that memory optimization is disabled.

Default value: off

approx_count_distinct_precision

Parameter description: Specifies the number of buckets in the HyperLogLog++ (HLL++) algorithm. This parameter can be used to adjust the error rate of the **approx_count_distinct** aggregate function. The number of buckets affects the precision of estimating the distinct value. Having more buckets increases the accuracy of the estimation. The deviation range is as follows: $[-1.04/2^{\log_2 m^{1/2}}, +1.04/2^{\log_2 m^{1/2}}]$

Type: USERSET

Value range: an integer ranging from 10 to 20.

Default value: 17

udf_memory_limit

Parameter description: Controls the maximum physical memory that can be used when each CN or DN executes UDFs.

Type: POSTMASTER

Value range: an integer ranging from 200 x 1024 to the value of [max_process_memory](#) and the unit is KB.

Default value: 0.05 * max_process_memory

FencedUDFMemoryLimit

Parameter description: Controls the virtual memory used by each fenced udf worker process.

Type: USERSET

Suggestion: You are not advised to set this parameter. You can set [udf_memory_limit](#) instead.

Value range: an integer. The unit can be KB, MB, or GB. 0 indicates that the memory is not limited.

Default value: 0

enable_pbe_optimization

Parameter description: Specifies whether the optimizer optimizes the query plan for statements executed in Parse Bind Execute (PBE) mode.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimizer optimizes the query plan.
- **off** indicates that the optimization does not optimize the query plan.

Default value: on

enable_light_proxy

Parameter description: Specifies whether the optimizer optimizes the execution of simple queries on CNs.

Type: USERSET

Value range: Boolean

- **on** indicates that the optimizer optimizes the execution.
- **off** indicates that the optimization does not optimize the execution.

Default value: on

enable_parallel_ddl

Parameter description: Controls whether multiple CNs can concurrently perform DDL operations on the same database object.

Type: USERSET

Value range: Boolean

- **on** indicates that DDL operations can be performed safely and that no distributed deadlock occurs.
- **off** indicates that DDL operations cannot be performed safely and that distributed deadlocks may occur.

Default value: on

gc_fdw_verify_option

Parameter description: Specifies whether to enable the logic for verifying the number of rows in a result set in the collaborative analysis. This parameter is supported only by clusters of version 8.1.3.310 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the logic for verifying the number of rows in the result set is enabled. The **SELECT COUNT** statement is used to obtain the expected number of rows and compare it with the actual number of rows.

- **off** indicates that the logic for verifying the number of rows in the result set is disabled and only the required result set is obtained.

Default value: on

 NOTE

- If this parameter is enabled, the performance deteriorates slightly. In performance-sensitive scenarios, you can disable this parameter to improve the performance.
- If the result set row count check fails, an exception will be reported. To enable cooperative analysis logs, set **log_min_messages** to **debug1** and **logging_module** to **'on(COOP_ANALYZE)'**.

show_acce_estimate_detail

Parameter description: When the DWS cluster is accelerated ([acceleration_with_compute_pool](#) is set to **on**), specifies whether the **EXPLAIN** statement displays the evaluation information about execution plan pushdown to computing Node Groups. The evaluation information is generally used by O&M personnel during maintenance, and it may affect the output display of the **EXPLAIN** statement. Therefore, this parameter is disabled by default. The evaluation information is displayed only if the **verbose** option of the **EXPLAIN** statement is enabled.

Type: USERSET

Value range: Boolean

- **on** indicates that the evaluation information is displayed in the output of the **EXPLAIN** statement.
- **off** indicates that the evaluation information is not displayed in the output of the **EXPLAIN** statement.

Default value: off

full_group_by_mode

Parameter description: Used in conjunction with **disable_full_group_by_mysql** in [behavior_compat_options](#) to control two different behaviors when **disable_full_group_by_mysql syntax** is enabled.

Type: USERSET

Value range: a string

- **nullpadding** indicates that NULL values in non-aggregate columns are filled with the non-NULL values in that column, potentially resulting in different rows in the result set.
- **notpadding** indicates that NULL values in non-aggregate columns are not processed, and the entire row data is used, resulting in a random row for non-aggregate columns in the result set.

Default value: notpadding

NOTICE

This parameter only takes effect when **disable_full_group_by_mysql** is enabled in the MySQL-compatible library and non-aggregate columns are present in the query. The two behaviors of this parameter only apply to non-aggregate columns in the query.

enable_cudesc_streaming

Parameter description: Specifies whether to use the cudesc streaming path for accessing data across logical clusters in the decoupled storage and compute architecture. This parameter is supported only by clusters of version 9.1.0 or later.

Type: SUSET

Value range: enumerated values

- **off** indicates that cudesc streaming is disabled.
- **on** indicates that cudesc streaming is enabled.
- **only_read_on** indicates that cudesc streaming is supported only during data reading.

Default value: on

force_read_from_rw

Parameter description: Forces data to be read from other logical clusters in the decoupled storage and compute architecture (i.e., read data from the logical cluster where the table resides). This parameter is supported only by clusters of version 9.0.0 or later.

Type: USERSET

Value range: Boolean

Default value: off

kv_sync_up_timeout

Parameter description: Specifies the timeout interval for KV synchronization in the decoupled storage and compute architecture. This parameter is supported only by clusters of version 9.0.0 or later.

Type: USERSET

Value range: an integer ranging from 0 to 2147483647

Default value: 10min

enable_insert_foreign_table_dop

Parameter description: Specifies whether to enable DOP acceleration when data is written into an OBS foreign table. The number of DOP threads on each DN is determined by **query_dop**. By adjusting its value, you can control the level of parallelism for your queries. This parameter is supported only in 9.1.0.200 and later versions.

Type: USERSET

Value range: Boolean

- **on** indicates that foreign table DOP acceleration is enabled.
- **off** indicates that foreign table DOP acceleration is disabled.

Default value: off

enable_insert_foreign_table_dop_opt

Parameter description: Specifies whether to enable partition redistribution optimization after **insert dop** is enabled for a foreign table. If the number of partitions to be exported is greater than 10 times the number of partitions, you are advised to enable this function to reduce small files in a single partition and improve export performance. This parameter is supported only by 9.1.0.200 and later versions.

Type: USERSET

Value range: Boolean

- **on** indicates that the **insert dop** redistribution optimization of the partitioned foreign table is enabled.
- **off** indicates that the insert dop redistribution optimization of the partitioned foreign table is disabled.

Default value: off

enable_hstore_binlog_table

Parameter description: Specifies whether binlog tables can be created.

Type: SIGHUP

Value range: Boolean

- **on** indicates that binlog tables can be created.
- **off** indicates that binlog tables cannot be created.

Default value: off

enable_generate_binlog

Parameter description: Specifies whether binlogs are generated for DML operations on binlog tables in the current session. This parameter is available only for clusters of version 9.1.0.200 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that binlogs are generated.
- **off** indicates that binlogs are not generated.

Default value: on

binlog_consume_timeout

Parameter description: Specifies the duration for cyclically determining whether all binlog records are consumed during binlog table scaling or **VACUUM FULL** operations. This parameter is available only for clusters of version 8.3.0.100 or later. The unit is second.

Type: SIGHUP

Value range: an integer ranging from 0 to 86400

Default value: 3600

19.21 Auditing

19.21.1 Audit Switch

audit_enabled

Parameter description: Specifies whether to enable or disable the audit process. After the audit process is enabled, the auditing information written by the background process can be read from the pipe and written into audit files.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the auditing function is enabled.
- **off** indicates that the auditing function is disabled.

Default value: on

audit_space_limit

Parameter description: Specifies the total disk space occupied by audit files.

Type: SIGHUP

Value range: an integer ranging from **1024** to **1073741824**. The unit is KB.

Default value: 1GB

audit_object_name_format

Parameter description: Specifies the format of the object name displayed in the **object_name** field of audit logs.

Type: USERSET

Value range: enumerated values

- **single** indicates that the **object_name** field displays a single object name, which is the name of the target object.
- **all** indicates that the **object_name** field displays multiple object names.

Default value: single

 NOTE

If the default value is set to **all**, multiple object names are displayed for SELECT, DELETE, UPDATE, INSERT, MERGE, CREATE TABLE AS, CREATE VIEW AS, DROP USER... CASCADE, DROP OWNED BY... CASCADE, DROP SCHEMA... CASCADE, DROP TABLE... CASCADE, DROP FOREIGN TABLE... CASCADE, and DROP VIEW... CASCADE.

audit_object_details

Parameter description: whether to record the **object_details** field in audit logs. This field indicates the table name, column name, and column type in the audit statement. This parameter is supported only by clusters of version 8.2.1.100 or later.

Type: USERSET

Value range: Boolean

- **on** indicates that the **object_details** field is recorded during the audit.
- **off** indicates that the **object_details** field is not recorded during the audit.

Default value: off

 NOTE

- If this parameter is set to **on**, the table name, column name, and column type in the statement will be audited, which may affect the performance. So, exercise caution when setting this parameter to **on**.
- If this parameter is set to **on**, the **object_details** field records the following statements: **SELECT**, **DELETE**, **UPDATE**, **INSERT**, **MERGE**, **CREATE TABLE AS SELECT**, **GRANT**, and **DECLARE CURSOR**. **GRANT** statements that fail to be executed are not recorded.

19.21.2 Operation Audit

security_enable_options

Parameter description: This parameter lets you use the **grant_to_public**, **grant_with_grant_option**, and **foreign_table_options** functions. These functions are off by default for security. Set this parameter as needed. It works with clusters version 8.2.0 and later.

Type: SIGHUP

Value range: a string

- **grant_to_public** indicates that **grant to public** can be used.
- **grant_with_grant_option** indicates that **with grant option** can be used.
- **foreign_table_options** allows users to perform operations on foreign tables without explicitly granting the **useft** permission to users.

Default value: empty

 NOTE

- In a newly installed cluster, this parameter is left blank by default, indicating that none of **grant_to_public**, **grant_with_grant_option**, and **foreign_table_options** can be used.
- In upgrade scenarios, the default value of this parameter is forward compatible. If the default values of **enable_grant_public** and **enable_grant_option** are **ON** before the upgrade, the default value of **security_enable_options** is **grant_to_public**, **grant_with_grant_option** after the upgrade.

19.22 Transaction Monitoring

By setting transaction timeout alerts, you can monitor transactions that are automatically rolled back and identify statement issues, as well as monitor statements that take too long to execute.

transaction_sync_naptime

Parameter description: For data consistency, when the local transaction's status differs from that in the snapshot of the GTM, other transactions will be blocked. You need to wait for a few minutes until the transaction status of the local host is consistent with that of the GTM. The **gs_clean** tool is automatically triggered for cleansing when the waiting period on the CN exceeds that of **transaction_sync_naptime**. The tool will shorten the blocking time after it completes the cleansing.

Type: USERSET

Value range: an integer. The minimum value is **0**. The unit is second.

Default value: 5s

 NOTE

If the value of this parameter is set to **0**, **gs_clean** will not be automatically invoked for the cleansing before the blocking arrives the duration. Instead, the **gs_clean** tool is invoked by **gs_clean_timeout**. The default value is 5 minutes.

transaction_sync_timeout

Parameter description: For data consistency, when the local transaction's status differs from that in the snapshot of the GTM, other transactions will be blocked. You need to wait for a few minutes until the transaction status of the local host is consistent with that of the GTM. An exception is reported when the waiting duration on the CN exceeds the value of **transaction_sync_timeout**. Roll back the transaction to avoid system blocking due to long time of process response failures (for example, sync lock).

Type: USERSET

Value range: an integer. The minimum value is **0**. The unit is second.

Default value: 10min

 NOTE

- If the value is **0**, no error is reported when the blocking times out or the transaction is rolled back.
- The value of this parameter must be greater than **gs_clean_timeout**. Otherwise, unnecessary transaction rollback will probably occur due to a block timeout caused by residual transactions that have not been deleted by **gs_clean** on a DN.

19.23 GTM Parameters

log_min_messages

Parameter description: Specifies which level of messages will be written into server logs. Each level covers all the levels following it. The lower the level is, the fewer messages will be written into the log.

NOTICE

If the values of **client_min_messages** and **log_min_messages** are the same, they indicate different levels.

Type: SUSET

Valid values: enumerated values. Valid values are **debug**, **debug5**, **debug4**, **debug3**, **debug2**, **debug1**, **info**, **log**, **notice**, **warning**, **error**, **fatal**, and **panic**. For details about the parameters, see [Table 19-1](#).

Default value: warning

19.24 Miscellaneous Parameters

server_version

Parameter description: Specifies the server version number in the string format.

Type: INTERNAL ((Fixed parameter. You are not advised to configure this parameter because incorrect values may cause compatibility issues.)

Value range: a string

Default value: 9.2.4

server_version_num

Parameter description: Specifies the server version number in the integer format.

Type: INTERNAL ((Fixed parameter. You are not advised to configure this parameter because incorrect values may cause compatibility issues.)

Value range: an integer

Default value: 90204

enable_cluster_resize

Parameter description: Indicates whether the current session is for scaling or redistributing data. It should only be used for these specific sessions and not set for other service sessions.

Type: SUSET

Value range: Boolean

- **on** indicates that the current session is for scaling or redistributing data, and allows the execution of specific SQL statements for redistribution.
- **off** indicates that the current session is not for scaling or redistributing data, and does not allow the execution of specific SQL statements for redistribution.

Default value: off

NOTE

This parameter is used for internal O&M. Do not set it to **on** unless absolutely necessary.

dfs_partition_directory_length

Parameter description: Specifies the largest directory name length for the partition directory of a table partitioned by VALUE in the HDFS.

Type: USERSET

Value range: 92 to 7999

Default value: 512

enable_hadoop_env

Parameter description: Sets whether local row- and column-store tables can be created in a database while the Hadoop feature is used. In the DWS cluster, it is set to **off** by default to support local row- and column- based storage and cross-cluster access to Hadoop. You are not advised to change the value of this parameter.

Type: USERSET

Value range: Boolean

- **on** or **true**, indicating that local row- and column-store tables cannot be created in a database while the Hadoop feature is used.
- **off** or **false**, indicating that local row- and column-based tables can be created in a database while the Hadoop feature is used.

Default value: off

enable_upgrade_merge_lock_mode

Parameter description: If this parameter is set to **on**, the delta merge operation internally increases the lock level, and errors can be avoided when update and delete operations are performed at the same time.

Type: USERSET

Value range: Boolean

- If this parameter is set to **on**, the delta merge operation internally increases the lock level. In this way, when any two of the **DELTAMERGE**, **UPDATE**, and **DELETE** operations are concurrently performed, an operation can be performed only after the previous one is complete.
- If this parameter is set to **off**, and any two of the **DELTAMERGE**, **UPDATE**, and **DELETE** operations are concurrently performed to data in a row in the delta table of the HDFS table, errors will be reported during the later operation, and the operation will stop.

Default value: off

job_queue_processes

Parameter description: Specifies the number of jobs that can be concurrently executed.

Type: POSTMASTER

Value range: 0 to 1000

Functions:

- Setting **job_queue_processes** to **0** indicates that the scheduled task function is disabled and that no job will be executed. (Enabling scheduled tasks may affect the system performance. At sites where this function is not required, you are advised to disable it.)
- Setting **job_queue_processes** to a value that is greater than **0** indicates that the scheduled task function is enabled and this value is the maximum number of tasks that can be concurrently processed.

After the scheduled task function is enabled, the **job_scheduler** thread at a scheduled interval polls the **pg_jobs** system catalog. The scheduled task check is performed every second by default.

Too many concurrent tasks consume many system resources, so you need to set the number of concurrent tasks to be processed. If the current number of concurrent tasks reaches **job_queue_processes** and some of them expire, these tasks 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 task to avoid the problem that tasks in the next polling period cannot be properly processed because overlong task execution time.

Note: If the number of parallel jobs is large and the value is too small, these jobs will wait in queues. However, a large parameter value leads to large resource consumption. You are advised to set this parameter to **100** and change it based on the system resource condition.

Default value: 10

ngram_gram_size

Parameter description: Specifies the length of the ngram parser segmentation.

Type: USERSET

Value range: an integer ranging from 1 to 4

Default value: 2

nggram_grapsymbol_ignore

Parameter description: Specifies whether the ngram parser ignores graphical characters.

Type: USERSET

Value range: Boolean

- **on:** Ignores graphical characters.
- **off:** Does not ignore graphical characters.

Default value: off

nggram_punctuation_ignore

Parameter description: Specifies whether the ngram parser ignores punctuations.

Type: USERSET

Value range: Boolean

- **on:** Ignores punctuations.
- **off:** Does not ignore punctuations.

Default value: on

zhparser_multi_duality

Parameter description: Specifies whether Zhparser aggregates segments in long words with duality.

Type: USERSET

Value range: Boolean

- **on:** Aggregates segments in long words with duality.
- **off:** Does not aggregate segments in long words with duality.

Default value: off

zhparser_multi_short

Parameter description: Specifies whether Zhparser executes long words composite divide.

Type: USERSET

Value range: Boolean

- **on:** Performs compound segmentation for long words.
- **off:** Does not perform compound segmentation for long words.

Default value: on

zhparser_multi_zall

Parameter description: Specifies whether Zhparser displays all single words individually.

Type: USERSET

Value range: Boolean

- **on:** Displays all single words separately.
- **off:** Does not display all single words separately.

Default value: off

zhparser_multi_zmain

Parameter description: Specifies whether Zhparser displays important single words separately.

Type: USERSET

Value range: Boolean

- **on:** Displays important single words separately.
- **off:** Does not display important single words separately.

Default value: off

zhparser_punctuation_ignore

Parameter description: Specifies whether the Zhparser segmentation result ignores special characters including punctuations (\r and \n will not be ignored).

Type: USERSET

Value range: Boolean

- **on:** Ignores all the special characters including punctuations.
- **off:** Does not ignore all the special characters including punctuations.

Default value: on

zhparser_seg_with_duality

Parameter description: Specifies whether Zhparser aggregates segments in long words with duality.

Type: USERSET

Value range: Boolean

- **on:** Aggregates segments in long words with duality.
- **off:** Does not aggregate segments in long words with duality.

Default value: off

acceleration_with_compute_pool

Parameter description: Specifies whether to use the computing resource pool for acceleration when OBS is queried.

Type: USERSET

Value range: Boolean

- **on** indicates that the query covering OBS is accelerated based on the cost when the computing resource pool is available.
- **off** indicates that no query is accelerated using the computing resource pool.

Default value: off

redact_compat_options

Parameter description: Specifies the compatibility option for calculation using masked data. This parameter is supported only by clusters of version 8.1.3.310 or later.

Type: USERSET

Value range: enumerated values

- **none** indicates that compatibility options are specified.
- **disable_comparison_operator_mask** indicates that comparison operators that do not expose raw data can bypass the data masking check and generate the actual calculation result.
- **relaxed_redact_check** indicates that the data masking column type verification is weakly controlled, and the table field column type that depends on data masking can be modified. This option is supported only by clusters of version 9.1.0.223 or later.

Default value: none

table_skewness_warning_threshold

Parameter description: Specifies the threshold for triggering a table skew alarm.

Type: SUSET

Value range: a floating point number ranging from 0 to 1

Default value: 1

table_skewness_warning_rows

Parameter description: Specifies the minimum number of rows for triggering a table skew alarm.

Type: SUSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 100000

enable_view_update

Parameter description: Enables the view update function or not.

Type: POSTMASTER

Value range: Boolean

- **on** indicates that the view update function is enabled.
- **off** indicates that the view update function is disabled.

Default value: off

view_independent

Parameter description: Decouples views from tables, functions, and synonyms or not. After the base table is restored, automatic association and re-creation are supported.

Type: SIGHUP

Value range: Boolean

- **on** indicates that the view decoupling function is enabled. Tables, functions, synonyms, and other views on which views depend can be deleted separately (except temporary tables and temporary views). Associated views are reserved but unavailable.
- **off** indicates that the view decoupling function is disabled. Tables, functions, synonyms, and other views on which views depend cannot be deleted separately. You can only delete them in the cascade mode.

Default value: off

assign_abort_xid

Parameter description: Determines the transaction to be aborted based on the specified XID in a query.

Type: USERSET

Value range: a character string with the specified XID



CAUTION

This parameter is used only for quick restoration if a user deletes data by mistake (DELETE operation). Do not use this parameter in other scenarios. Otherwise, visible transaction errors may occur.

default_distribution_mode

Parameter description: Specifies the default distribution mode of a table. This feature is supported only in 8.1.2 or later.

Type: USERSET

Value range: enumerated values

- **roundrobin**: If the distribution mode is not specified during table creation, the default distribution mode is selected according to the following rules:
 - a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - b. If the primary key or unique constraint is not included during table creation, round-robin distribution is selected.
- **hash**: If the distribution mode is not specified during table creation, the default distribution mode is selected according to the following rules:
 - a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - b. 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 columns, hash distribution is selected. The distribution column is the first column whose data type can be used as a distribution column.
 - c. 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 column exists, round-robin distribution is selected.

Default value: roundrobin

 NOTE

The default value of this parameter is **roundrobin** for a new DWS 8.1.2 cluster and is **hash** for an upgrade to DWS 8.1.2.

object_mtime_record_mode

Parameter description: Sets the update action of the **mtime** column in the **PG_OBJECT** system catalog.

Type: SIGHUP

Value range: a string

- **default**: **ALTER**, **COMMENT**, **GRANT/REVOKE**, and **TRUNCATE** operations update the **mtime** column by default.
- **disable**: The **mtime** column is not updated.
- **disable_acl**: **GRANT** or **REVOKE** operation does not update the **mtime** column.
- **disable_truncate**: **TRUNCATE** operations do not update the **mtime** column.
- **disable_partition**: Partition **ALTER** operations do not update the **mtime** column.

Default value: default

max_volatile_tables

Parameter description: Specifies the maximum number of volatile tables created for each session, including volatile tables and their auxiliary tables. This parameter is supported by clusters of version 8.2.0 or later.

Type: USERSET

Value range: an integer ranging from 0 to INT_MAX

Default value: 300

query_cache_refresh_time

Parameter description: Specifies the cache refresh interval for queries for which the **enable_accelerate_select** parameter takes effect. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: a floating point number ranging from 0 to 10000.0, in seconds

Default value: 60.0

vector_engine_strategy

Parameter description: Specifies the vectorization enhancement policy. This parameter is supported only by clusters of version 8.3.0 or later.

Type: USERSET

Value range: enumerated values

- **force** specifies that the vectorization-enhanced plan is forcibly rolled back to the row storage plan when there are scenarios that do not support vectorization.
- **improve** specifies that vectorization enhancement is enabled even when there are scenarios that do not support vectorization.

Default value: improve

default temptable_type

Parameter description: Specifies the type of temporary table created when **CREATE TABLE** is used to create a temporary table without specifying the table type before **TEMP** or **TEMPORARY**. This parameter is supported only by clusters of version 9.1.0 or later.

Type: USERSET

Value range: enumerated values

- **local**: creates a local temporary table when the type is not specified.
- **volatile**: creates a volatile temporary table when the type is not specified.

Default value: local

pgxc_node_readonly

Parameter description: Specifies whether a CN or DN is an elastic or classic DN. This parameter is supported only by clusters of version 9.1.0 or later.

Type: SUSET

Value range: Boolean

- **on** indicates that the CN or DN is an elastic node.
- **off** indicates that the CN or DN is a classic node.

Default value: off

hudi_sync_max_commits

Parameter description: Specifies the maximum number of commits for a single synchronization task in Hudi. This parameter is supported only by clusters of version 9.1.0.100 or later.

Type: SIGHUP

Value range: an integer ranging from -1 to INT_MAX

- **-1** indicates no limit.
- **0** indicates no limit.
- Any other value indicates the maximum number of commits.

Default value: -1

foreign_table_default_rw_options

Parameter description: Specifies the default permissions when creating a foreign table without specifying them. This parameter is supported only by clusters of version 9.0.3 or later.

Type: USERSET

Value range: a string

- **READ_ONLY** indicates the read-only permission.
- **WRITE_ONLY** indicates the write-only permission.
- **READ_WRITE** indicates the read-write permission.

Default value: READ_ONLY

20 DWS Developer Terms

A~E

Table 20-1 Terms A to E

| Term | Description |
|--------------|--|
| ACID | Four essential properties that a transaction should have in a DBMS: Atomicity, Consistency, Isolation, and Durability. |
| cluster ring | A cluster ring consists of several physical servers. The primary-standby-secondary relationships among its DNs do not involve external DNs. That is, none of the primary, standby, or secondary counterparts of DNs belonging to the ring are deployed in other rings. A ring is the smallest unit used for scaling. |
| Bgwriter | A background write thread created when the database starts. The thread pushes dirty pages in the database to a permanent device (such as a disk). |
| bit | The smallest unit of information handled by a computer. One bit is expressed as a 1 or a 0 in a binary numeral, or as a true or a false logical condition. A bit is physically represented by an element such as high or low voltage at one point in a circuit, or a small spot on a disk that is magnetized in one way or the other. A single bit conveys little information a human would consider meaningful. A group of eight bits, however, makes up a byte, which can be used to represent many types of information, such as a letter of the alphabet, a decimal digit, or other character. |
| Bloom filter | Bloom filter is a space-efficient binary vectorized data structure, conceived by Burton Howard Bloom in 1970, that is used to test whether an element is a member of a set. False positive matches are possible, but false negatives are not, in other words, a query returns either "possibly in set (possible error)" or "definitely not in set". In the cases, Bloom filter sacrificed the accuracy for time and space. |

| Term | Description |
|---------|---|
| CCN | The Central Coordinator (CCN) is a node responsible for determining, queuing, and scheduling complex operations in each CN to enable the dynamic load management of DWS. |
| CIDR | Classless Inter-Domain Routing (CIDR). CIDR abandons the traditional class-based (class A: 8; class B: 16; and class C: 24) address allocation mode and allows the use of address prefixes of any length, effectively improving the utilization of address space. A CIDR address is in the format of <i>IP address/Number of bits in a network ID</i> . For example, in 192.168.23.35/21, 21 indicates that the first 21 bits are the network prefix and others are the host ID. |
| Cgroups | A control group (Cgroup), also called a priority group (PG) in DWS. The Cgroup is a kernel feature of SUSE Linux and Red Hat that can limit, account for, and isolate the resource usage of a collection of processes. |
| CLI | Command-line interface (CLI). Users use the CLI to interact with applications. Its input and output are based on texts. Commands are entered through keyboards or similar devices and are compiled and executed by applications. The results are displayed in text or graphic forms on the terminal interface. |
| CM | Cluster Manager (CM) manages and monitors the running status of functional units and physical resources in the distributed system, ensuring stable running of the entire system. |
| CMS | The Cluster Management Service (CMS) component manages the cluster status. |
| CN | The Coordinator (CN) stores database metadata, splits query tasks and supports their execution, and aggregates the query results returned from DNs. |
| CTE | A common table expression (CTE) is a named temporary result set that exists within the scope of a single statement and that can be referred to later in SELECT, INSERT, UPDATE, or DELETE statements multiple times. It simplifies complex queries, and enhances code readability and maintainability. CTEs can significantly improve the maintainability of complex SQL statements, and have obvious advantages in recursive or step-by-step data processing. |
| CU | A compression unit is the smallest storage unit of a column-store table. |

| Term | Description |
|-----------|---|
| core file | <p>A file that is created when memory overwriting, assertion failures, or access to invalid memory occurs in a process, causing it to fail. This file is then used for further analysis.</p> <p>A core file contains a memory dump, in an all-binary and port-specific format. The name of a core file consists of the word "core" and the OS process ID.</p> <p>The core file is available regardless of the type of platform.</p> |
| core dump | <p>When a program stops abnormally, the core dump, memory dump, or system dump records the state of the working memory of the program at that point in time. In practice, other key pieces of program state are usually dumped at the same time, including the processor registers, which may include the program counter and stack pointer, memory management information, and other processor and OS flags and information. A core dump is often used to assist diagnosis and computer program debugging.</p> |
| DBA | <p>A database administrator (DBA) instructs or executes database maintenance operations.</p> |
| DBLINK | <p>An object defining the path from one database to another. A remote database object can be queried with DBLINK.</p> |
| DBMS | <p>Database Management System (DBMS) is a piece of system management software that allows users to access information in a database. This is a collection of programs that allows you to access, manage, and query data in a database. A DBMS can be classified as memory DBMS or disk DBMS based on the location of the data.</p> |
| DCL | <p>Data control language (DCL)</p> |
| DDL | <p>Data definition language (DDL)</p> |
| DFS | <p>A Distributed File System (DFS) stores files on multiple physical nodes and allows access to the files via unified access interfaces. It is not a specific technique, but a system type.</p> <p>For example, Hadoop Distributed File System (HDFS) is a type of DFS and is designed for big data scenarios. It is suitable for high-throughput and batch processing tasks.</p> |
| DML | <p>Data manipulation language (DML)</p> |
| DN | <p>Datanode performs table data storage and query operations.</p> |
| ETCD | <p>The Editable Text Configuration Daemon (ETCD) is a distributed key-value storage system used for configuration sharing and service discovery (registration and search).</p> |
| ETL | <p>Extract-Transform-Load (ETL) refers to the process of data transmission from the source to the target database.</p> |

| Term | Description |
|------------------------|---|
| Extension Connector | Extension Connector is provided by DWS to process data across clusters. It can send SQL statements to Spark, and can return execution results to your database. |
| Backup | A backup, or the process of backing up, refers to the copying and archiving of computer data in case of data loss. |
| backup and restoration | A collection of concepts, procedures, and strategies to protect data loss caused by invalid media or misoperations. |
| standby server | A node in the DWS HA solution. It functions as a backup of the primary server. If the primary server is behaving abnormally, the standby server is promoted to primary, ensuring data service continuity. |
| crash | A crash (or system crash) is an event in which a computer or a program (such as a software application or an OS) ceases to function properly. Often the program will exit after encountering this type of error. Sometimes the offending program may appear to freeze or hang until a crash reporting service documents details of the crash. If the program is a critical part of the OS kernel, the entire computer may crash (possibly resulting in a fatal system error). |
| encoding | Encoding is representing data and information using code so that it can be processed and analyzed by a computer. Characters, digits, and other objects can be converted into digital code, or information and data can be converted into the required electrical pulse signals based on predefined rules. |
| encoding technology | A technology that presents data using a specific set of characters, which can be identified by computer hardware and software. |
| table | A set of columns and rows. Each column is referred to as a field. The value in each field represents a data type. For example, if a table contains people's names, cities, and states, it has three columns: Name , City , and State . In every row in the table, the Name column contains a name, the City column contains a city, and the State column contains a state. |
| tablespace | A tablespace is a logical storage structure that contains tables, indexes, large objects, and long data. A tablespace provides an abstract layer between physical data and logical data, and provides storage space for all database objects. When you create a table, you can specify which tablespace it belongs to. |
| concurrency control | A DBMS service that ensures data integrity when multiple transactions are concurrently executed in a multi-user environment. In a multi-threaded environment, DWS concurrency control ensures that database operations are safe and all database transactions remain consistent at any given time. |

| Term | Description |
|------------------------|---|
| query | A request sent to a database, which may include updates, modifications, retrieval, or deletions of information. |
| query operator | An iterator or a query tree node, which is a basic unit for the execution of a query. Execution of a query can be split into one or more query operators. Common query operators include scan, join, and aggregation. |
| query fragment | Each query task can be split into one or more query fragments. Each query fragment consists of one or more query operators and can independently run on a node. Query fragments exchange data through data flow operators. |
| durability | One of the ACID features of database transactions. Durability indicates that transactions that have been committed will permanently survive and not be rolled back. |
| stored procedure | A group of SQL statements compiled into a single execution plan and stored in a large database system. Users can specify a name and parameters (if any) for a stored procedure to execute the procedure. |
| OS | An operating system (OS) is loaded by a bootstrap program to a computer to manage other programs in the computer. Other programs are applications or application programs. |
| secondary server | To ensure high cluster availability, the primary server synchronizes logs to the secondary server if data synchronization between the primary and standby servers fails. If the primary server suddenly breaks down, the standby server is promoted to primary and synchronizes logs from the secondary server for the duration of the breakdown. |
| BLOB | Binary large object (BLOB) is a collection of binary data stored in a database, such as videos, audio, and images. |
| recursive query | Recursive query is an advanced technology in database query. It is mainly used to process hierarchical data or used in scenarios where layer-by-layer iterative calculation is required. It traverses and analyzes tree structures, graph structures, and chain relationships through recursive invocation. In the SELECT syntax, WITH RECURSIVE is often used to declare recursive queries. For the examples of using recursive queries, see "SELECT" in SQL Syntax. |
| dynamic load balancing | In DWS, dynamic load balancing automatically adjusts the number of concurrent jobs based on the usage of CPU, I/O, and memory to avoid service errors and to prevent the system from stop responding due to system overload. |

| Term | Description |
|---------|--|
| segment | A segment in the database indicates a part containing one or more regions. Region is the smallest range of a database and consists of data blocks. One or more segments comprise a tablespace. |

F~J**Table 20-2** Terms F to J

| Term | Description |
|----------|---|
| Failover | Automatic switchover from a faulty node to its standby node. Reversely, automatic switchback from the standby node to the primary node is called failback. |
| FDW | A foreign data wrapper (FDW) is a SQL interface provided by Postgres. It is used to access big data objects stored in remote data so that DBAs can integrate data from unrelated data sources and store them in public schema in the database. |
| freeze | An operation automatically performed by the AutoVacuum Worker process when transaction IDs are exhausted. DWS records transaction IDs in row headings. When a transaction reads a row, the transaction ID in the row heading and the actual transaction ID are compared to determine whether this row is explicit. Transaction IDs are integers containing no symbols. If exhausted, transaction IDs are re-calculated outside of the integer range, causing the explicit rows to become implicit. To prevent such a problem, the freeze operation marks a transaction ID as a special ID. Rows marked with these special transaction IDs are explicit to all transactions. |
| GDB | As a GNU debugger, GDB allows you to see what is going on 'inside' another program while it executes or what another program was doing the moment that it crashed. GDB can perform four main kinds of things (make PDK functions stronger) to help you catch bugs in the act: <ul style="list-style-type: none">• Starts your program, specifying anything that might affect its behavior.• Stops a program in a specific condition.• Checks what happens when a program stops.• Modifies the program content to rectify the fault and proceeds with the next one. |
| GDS | General Data Service (GDS). To import data to DWS, you need to deploy the tool on the server where the source data is stored so that DNs can use this tool to obtain data. |

| Term | Description |
|-----------|--|
| GIN index | Generalized inverted index (GIN) is used for handling cases where the items to be indexed are composite values, and the queries to be handled by the index need to search for element values that appear within the composite items. |
| GNU | The GNU Project was publicly announced on September 27, 1983 by Richard Stallman, aiming at building an OS composed wholly of free software. GNU is a recursive acronym for "GNU's Not Unix!". Stallman announced that GNU should be pronounced as Guh-NOO. Technically, GNU is similar to Unix in design, a widely used commercial OS. However, GNU is free software and contains no Unix code. |
| gsql | DWS interaction terminal. It enables you to interactively type in queries, issue them to DWS, and view the query results. Queries can also be entered from files. gsql supports many meta commands and shell-like commands, allowing you to conveniently compile scripts and automate tasks. |
| GTM | Global Transaction Manager (GTM) manages the status of transactions. |
| GUC | Grand unified configuration (GUC) includes parameters for running databases, the values of which determine database system behavior. |
| HA | High availability (HA) is a solution in which two modules operate in primary/standby mode to achieve high availability. This solution helps to minimize the duration of service interruptions caused by routine maintenance (planned) or sudden system breakdowns (unplanned), improving the system and application usability. |
| HBA | Host-based authentication (HBA) allows hosts to authenticate on behalf of all or some of the system users. It can apply to all users on a system or a subset using the Match directive. This type of authentication can be useful for managing computing clusters and other fairly homogenous pools of machines. In all, three files on the server and one on the client must be modified to prepare for host-based authentication. |
| HDFS | Hadoop Distributed File System (HDFS) is a subproject of Apache Hadoop. HDFS is highly fault tolerant and is designed to run on low-end hardware. The HDFS provides high-throughput access to large data sets and is ideal for applications having large data sets. |
| server | A combination of hardware and software designed for providing clients with services. This word alone refers to the computer running the server OS, or the software or dedicated hardware providing services. |

| Term | Description |
|----------------------|---|
| advanced package | Logical and functional stored procedures and functions provided by DWS. |
| isolation | One of the ACID features of database transactions. Isolation means that the operations inside a transaction and data used are isolated from other concurrent transactions. The concurrent transactions do not affect each other. |
| relational database | A database created using a relational model. It processes data using methods of set algebra. |
| archive thread | A thread started when the archive function is enabled on a database. The thread archives database logs to a specified path. |
| failover | The automatic substitution of a functionally equivalent system component for a failed one. The system component can be a processor, server, network, or database. |
| environment variable | An environment variable defines the part of the environment in which a process runs. For example, it can define the part of the environment as the main directory, command search path, terminal that is in use, or the current time zone. |
| checkpoint | A mechanism that stores data in the database memory to disks at a certain time. DWS periodically stores the data of committed and uncommitted transactions to disks. The data and redo logs can be used for database restoration if a database restarts or breaks down. |
| encryption | A function hiding information content during data transmission to prevent the unauthorized use of the information. |
| node | Cluster nodes (or nodes) are physical and virtual servers that make up the DWS cluster environment. |
| error correction | A technique that automatically detects and corrects errors in software and data streams to improve system stability and reliability |
| process | An instance of a computer program that is being executed. A process may be made up of multiple threads of execution. Other processes cannot use a thread occupied by the process. |
| PITR | Point-In-Time Recovery (PITR) is a backup and restoration feature of DWS. Data can be restored to a specified point in time if backup data and WALs are normal. |
| record | In a relational database, a record corresponds to data in each row of a table. |
| cluster | A cluster is an independent system consisting of servers and other resources, ensuring high availability. In certain conditions, clusters can implement load balancing and concurrent processing of transactions. |

K~O

Table 20-3 Terms K to O

| Term | Description |
|--------------------------|--|
| LLVM | <p>LLVM is short for Low Level Virtual Machine. Low Level Virtual Machine (LLVM) is a compiler framework written in C++ and is designed to optimize the compile-time, link-time, run-time, and idle-time of programs that are written in arbitrary programming languages. It is open to developers and compatible with existing scripts.</p> <p>DWS LLVM dynamic compilation can be used to generate customized machine code for each query to replace original common functions. Query performance is improved by reducing redundant judgment conditions and virtual function invocation, and by making local data more accurate during actual queries.</p> |
| LVS | Linux Virtual Server (LVS), a virtual server cluster system, is used for balancing the load of a cluster. |
| logical replication | Data synchronization mode between primary and standby databases or between two clusters. Different from physical replication which replays physical logs, logical replication transfers logical logs between two clusters or synchronizes data through SQL statements in logical logs. |
| logical log | Logs recording database changes made through SQL statements. Generally, the changes are logged at the row level. Logical logs are different from physical logs that record changes of physical pages. |
| logical decoding | Logic decoding is a process of extracting all permanent changes in database tables into a clear and easy-to-understand format by decoding Xlogs. |
| logical replication slot | In a logical replication process, logic replication slots are used to prevent Xlogs from being reclaimed by the system or VACUUM . In DWS, a logical replication slot is an object that records logical decoding positions. It can be created, deleted, read, and pushed by invoking SQL functions. |
| MPP | Massive Parallel Processing (MPP) refers to cluster architecture that consists of multiple machines. The architecture is also called a cluster system. |
| MVCC | Multi-Version Concurrency Control (MVCC) is a protocol that allows a tuple to have multiple versions, on which different query operations can be performed. A basic advantage is that read and write operations do not conflict. |

| Term | Description |
|-----------------------|---|
| NameNode | The NameNode is the centerpiece of a Hadoop file system, managing the namespace of the file system and client access to files. |
| mmap | Memory map, that is, memory mapping. mmap maps files or other objects into the address space of a process. The files or devices can be accessed as if they were in the memory, without being read or written. mmap enables efficient file reading and writing and inter-process communications. |
| Node Group | In DWS, a Node Group refers to a DN set, which is a sub-cluster. Node Groups can be classified into Storage Node Groups, which store local table data; and Computing Node Groups, which perform aggregation and join for queries. |
| NULL FIRST | <p>In SQL, NULL FIRST is a clause used to explicitly control the positions of NULL values during sorting. It is usually used in the ORDER BY statement. It puts NULL values before non-NULL values.</p> <p>When NULL FIRST is used in ORDER BY, all rows with NULL values are displayed first in the result set, followed by the rows with non-null values, regardless of whether the sorting order is ASC or DESC.</p> |
| OLAP | Online analytical processing (OLAP) is the most important application in the database warehouse system. It is dedicated to complex analytical operations, helps decision makers and executives to make decisions, and rapidly and flexibly processes complex queries involving a great amount of data based on analysts' requirements. In addition, the OLAP provides decision makers with query results that are easy to understand, allowing them to learn the operating status of the enterprise. These decision makers can then produce informed and accurate solutions based on the query results. |
| OM | Operations Management (OM) provides management interfaces and tools for routine maintenance and configuration management of the cluster. |
| ORC | Optimized Row Columnar (ORC) is a widely used file format for structured data in a Hadoop system. It was introduced from the Hadoop HIVE project. |
| client | A computer or program that accesses or requests services from another computer or program. |
| free space management | A mechanism for managing free space in a table. This mechanism enables the database system to record free space in each table and establish an easy-to-search data structure, accelerating operations (such as INSERT) performed on the free space. |

| Term | Description |
|---------------|--|
| cross-cluster | In DWS, users can access data in other DBMS through foreign tables or using an Extension Connector. Such access is cross-cluster. |
| junk tuple | A tuple that is deleted using the DELETE and UPDATE statements. When deleting a tuple, DWS only marks the tuples that are to be cleared. The Vacuum thread will then periodically clear these junk tuples. |
| column | An equivalent concept of "field". A database table consists of one or more columns. Together they describe all attributes of a record in the table. |
| logical node | Multiple logical nodes can be installed on the same node. A logical node is a database instance. |
| schema | A collection of database objects that define the logical structure, such as tables, views, sequences, stored procedures, synonyms, indexes, clusters, and database links. |
| schema file | A SQL file that determines the database structure. |

P~T**Table 20-4** Terms P to T

| Term | Description |
|-------------|--|
| Page | Minimum memory unit for row storage in the DWS relational object structure. The default size of a page is 8 KB. By default, the page size is determined during database initialization and cannot be dynamically changed. |
| PostgreSQL | An open-source DBMS developed by volunteers all over the world. PostgreSQL is not controlled by any companies or individuals. Its source code can be used for free. |
| Postgres-XC | Postgres-XC is an open source PostgreSQL cluster to provide write-scalable, synchronous, multi-master PostgreSQL cluster solution. |
| Postmaster | A thread started when the database service is started. It listens to connection requests from other nodes in the cluster or from clients.

After receiving and accepting a connection request from the standby server, the primary server creates a WAL Sender thread to interact with the standby server. |
| RHEL | Red Hat Enterprise Linux (RHEL) |

| Term | Description |
|-----------------------------|--|
| redo log | A log that contains information required for performing an operation again in a database. If a database is faulty, redo logs can be used to restore the database to its original state. |
| SCTP | The Stream Control Transmission Protocol (SCTP) is a transport-layer protocol defined by Internet Engineering Task Force (IETF) in 2000. The protocol ensures the reliability of datagram transport based on unreliable service transmission protocols by transferring SCN narrowband signaling over IP network. |
| savepoint | A savepoint marks the end of a sub-transaction (also known as a nested transaction) in a relational DBMS. The process of a long transaction can be divided into several parts. After a part is successfully executed, a savepoint will be created. If later execution fails, the transaction will be rolled back to the savepoint instead of being totally rolled back. This is helpful for recovering database applications from complex errors. If an error occurs in a multi-statement transaction, the application can recover by rolling back to the save point without terminating the entire transaction. |
| session | A task created by a database for a connection when an application attempts to connect to the database. Sessions are managed by the session manager. They execute initial tasks to perform all user operations. |
| shared-nothing architecture | A distributed computing architecture, in which none of the nodes share CPUs or storage resources. This architecture has good scalability. |
| SIMD | Single Instruction, Multiple Data (SIMD) is a parallel computing technique where a single instruction operates on multiple data elements simultaneously. It significantly improves the performance of compute-intensive tasks. |
| SLES | SUSE Linux Enterprise Server (SLES) is an enterprise Linux OS provided by SUSE. |
| SMP | Symmetric multiprocessing (SMP) lets multiple CPUs run on a computer and share the same memory and bus. To ensure an SMP system achieves high performance, an OS must support multi-tasking and multi-thread processing. In databases, SMP means to concurrently execute queries using the multi-thread technology, efficiently using all CPU resources and improving query performance. |
| SQL | Structure Query Language (SQL) is a standard database query language. It consists of DDL, DML, and DCL. |

| Term | Description |
|----------------------|--|
| SSL | Secure Socket Layer (SSL) is a network security protocol introduced by Netscape. It is a security protocol based on the TCP and IP communications protocols and uses the public key technology. SSL supports a wide range of networks and provides three basic security services, all of which use the public key technology. SSL ensures the security of service communication through the network by establishing a secure connection between the client and server and then sending data through this connection. |
| convergence ratio | Downlink to uplink bandwidth ratio of a switch. A high convergence ratio indicates a highly converged traffic environment and severe packet loss. |
| TCP | Transmission Control Protocol (TCP) sends and receives data through the IP protocol. It splits data into packets for sending, and checks and reassembles received package to obtain original information. TCP is a connection-oriented, reliable protocol that ensures information correctness in transmission. |
| trace | A way of logging to record information about the way a program is executed. This information is typically used by programmers for debugging purposes. System administrators and technical support can diagnose common problems by using software monitoring tools and based on this information. |
| escape character | In a database, an escape character is a special character used to escape other characters. It enables the database system to identify and correctly process characters that have special meanings and may conflict with SQL syntax (such as quotation marks and special characters) as common characters. Common escape characters include backslashes (\) and double quotation marks ("). |
| full backup | Backup of the entire database cluster. |
| full synchronization | A data synchronization mechanism specified in the DWS HA solution. Used to synchronize all data from the primary server to a standby server. |
| log file | A file to which a computer system writes a record of its activities. |
| transaction | A logical unit of work performed within a DBMS against a database. A transaction consists of a limited database operation sequence, and must have ACID features. |
| data | A representation of facts or directives for manual or automatic communication, explanation, or processing. Data includes constants, variables, arrays, and strings. |
| data redistribution | A process whereby a data table is redistributed among nodes after users change the data distribution mode. |

| Term | Description |
|--------------------|---|
| data distribution | A mode in which table data is split and stored on each database instance in a distributed system. Table data can be distributed in hash, replication, or random mode. In hash mode, a hash value is calculated based on the value of a specified column in a tuple, and then the target storage location of the tuple is determined based on the mapping between nodes and hash values. In replication mode, tuples are replicated to all nodes. In random mode, data is randomly distributed to the nodes. |
| data partitioning | A division of a logical database or its constituent elements into multiple parts (partitions) whose data does not overlap based on specified ranges. Data is mapped to storage locations based on the value ranges of specific columns in a tuple. |
| database | A collection of data that is stored together and can be accessed, managed, and updated. Data in a view in the database can be classified into the following types: numerals, full text, digits, and images. |
| DB instance | A database instance consists of a process in DWS and files controlled by the process. DWS installs multiple database instances on one physical node. GTM, CM, CN, and DN installed on cluster nodes are all database instances. A database instance is also called a logical node. |
| database HA | DWS provides a highly reliable HA solution. Every logical node in DWS is identified as a primary or standby node. Only one DWS node is identified as primary at a time. When the HA system is deployed for the first time, the primary server synchronizes all data from each standby server (full synchronization). The HA system then synchronizes only data that is new or has been modified from each standby server (incremental synchronization). When the HA system is running, the primary server can receive data read and write operation requests and the standby servers only synchronize logs. |
| database file | A binary file that stores user data and the data inside the database system. |
| data flow operator | An operator that exchanges data among query fragments. By their input/output relationships, data flows can be categorized into Gather flows, Broadcast flows, and Redistribution flows. Gather combines multiple query fragments of data into one. Broadcast forwards the data of one query fragment to multiple query fragments. Redistribution reorganizes the data of multiple query fragments and then redistributes the reorganized data to multiple query fragments. |
| data dictionary | A reserved table within a database which is used to store information about the database itself. The information includes database design information, stored procedure information, user rights, user statistics, database process information, database increase statistics, and database performance statistics. |

| Term | Description |
|------------|--|
| deadlock | Unresolved contention for the use of resources. |
| index | An ordered data structure in the database management system. An index accelerates querying and the updating of data in database tables. |
| statistics | Information that is automatically collected by databases, including table-level information (number of tuples and number of pages) and column-level information (column value range distribution histogram). Statistics in databases are used to estimate the cost of execution plans to find the plan with the lowest cost. |
| stop word | In computing, stop words are words which are filtered out before or after processing of natural language data (text), saving storage space and improving search efficiency. |

U~Z

Table 20-5 Terms U to Z

| Term | Description |
|----------------|---|
| unlogged table | Unlogged tables are a special type of tables that do not record write-ahead logs (WALs) for data operations. These table can significantly improve write performance in some scenarios. However, they are automatically truncated upon conflicts, OS restart, forcible restart, power-off, or unexpected shutdown, which may cause data loss. The content of an unlogged table is not replicated to standby servers. Any indexes created on an unlogged table are not automatically logged as well.
The CREATE TABLE syntax creates unlogged tables if UNLOGGED is specified. |
| Vacuum | A thread that is periodically started up by a database to clear junk tuples. Multiple Vacuum threads can be started concurrently by setting a parameter. |
| V2 table | A V2 table refers to a table whose colversion defined in the CREATE TABLE syntax is 2.0 during table creation, indicating that each column of the column-store table is combined and stored in a file named relfilenode.C1.0 , and data is stored on the local disk. For storage-compute coupled clusters, if colversion is not specified, the created column-store table is a V2 table by default. |

| Term | Description |
|----------------|---|
| V3 table | A V3 table refers to a table whose colversion defined in the CREATE TABLE syntax is 3.0 during table creation, indicating that each column of the column-store table is stored in a file named C1_field.0 , and data is stored in the OBS file system. For storage-compute coupled clusters, if colversion is not specified, the created column-store table is a V3 table by default. |
| verbose | The VERBOSE option specifies the information to be displayed. |
| WAL | Write-ahead logging (WAL) is a standard method for logging a transaction. Corresponding logs must be written into a permanent device before a data file (carrier for a table and index) is modified. |
| WAL Receiver | A thread created by the standby server during database duplication. The thread is used to receive data and commands from the primary server and to tell the primary server that the data and commands have been acknowledged. Only one WAL receiver thread can run on one standby server. |
| WAL Sender | Name of a thread created by the primary node after the primary node receives a connection request from the standby node during database replication. This thread is used to send data and commands to standby servers and to receive responses from the standby servers. Multiple WAL Sender threads may run on one primary server. Each WAL Sender thread corresponds to a connection request initiated by a standby server. |
| WAL Writer | A thread for writing redo logs that are created when a database is started. This thread is used to write logs in the memory to a permanent device, such as a disk. |
| WLM | The WorkLoad Manager (WLM) is a module for controlling and allocating system resources in DWS. |
| Xlog | A transaction log. A logical node can have only one Xlog file. |
| xDR | X detailed record. It refers to detailed records on the user and signaling plans and can be categorized into charging data records (CDRs), user flow data records (UFDRs), transaction detail records (TDRs), and data records (SDRs). |
| network backup | Network backup provides a comprehensive and flexible data protection solution to Windows, UNIX, and Linux platforms. Network backup can back up, archive, and restore files, folders, directories, volumes, and partitions on a computer. |
| Predicate | A predicate is a logical expression used to filter or limit data. It is essentially a conditional expression that returns a Boolean value (TRUE , FALSE , or NULL). Predicates are used in WHERE, CHECK, and JOIN clauses. For example, salary > 5000 in WHERE salary > 5000 , is a predicate. |

| Term | Description |
|-------------------|---|
| predicate column | A predicate column is a column referenced in a predicate expression. For example, salary in salary > 5000 is the predicate column. If the predicate column in a query contains an index, the query optimizer may use the index to accelerate data filtering. This is called "index scan". DWS collects statistics on predicate columns (supported by clusters of version 9.1.0.100 and later) to help generate better query plans. |
| physical node | A physical machine or device. |
| materialized view | A materialized view is a special database object. It pre-computes complex query results and stores them in the database to accelerate queries. |
| system catalog | A table storing meta information about the database. The meta information includes user tables, indexes, columns, functions, and the data types in a database. |
| pushdown | DWS is a distributed database, where CN can send a query plan to multiple DNs for parallel execution. This CN behavior is called pushdown. It achieves better query performance than extracting data to CN for query. |
| compression | Data compression, source coding, or bit-rate reduction involves encoding information that uses fewer bits than the original representation. Compression can be either lossy or lossless. Lossless compression reduces bits by identifying and eliminating statistical redundancy. No information is lost in lossless compression. Lossy compression reduces bits by identifying and removing unnecessary or unimportant information. The process of reducing the size of a data file is commonly referred as data compression, although its formal name is source coding (coding done at the source of the data, before it is stored or transmitted). |
| consistency | One of the ACID features of database transactions. Consistency is a database status. In such a status, data in the database must comply with integrity constraints. |
| metadata | Data that provides information about other data. Metadata describes the source, size, format, or other characteristics of data. In database columns, metadata explains the content of a data warehouse. |
| atomicity | One of the ACID features of database transactions. Atomicity means that a transaction is composed of an indivisible unit of work. All operations performed in a transaction must either be committed or uncommitted. If an error occurs during transaction execution, the transaction is rolled back to the state when it was not committed. |

| Term | Description |
|-----------------------------|--|
| Zhparser | Zhparser is an extension for full-text search of Chinese. It provides Chinese word segmentation, enabling DWS to process Chinese text search and analysis requests more efficiently. |
| online scale-out | Online scale-out means that data can be saved to the database and query services are not interrupted during redistribution in DWS. |
| dirty page | A page that has been modified and is not written to a permanent device. |
| incremental backup | Incremental backup stores all files changed since the last valid backup. |
| incremental synchronization | A data synchronization mechanism in the DWS HA solution. Only data modified since the last synchronization is synchronized to the standby server. |
| Host | A node that receives data read and write operations in the DWS HA system and works with all standby servers. At any time, only one node in the HA system is identified as the primary server. |
| thesaurus | Standardized words or phrases that express document themes and are used for indexing and retrieval. |
| dump file | A specific type of the trace file. A dump is typically a one-time output of diagnostic data in response to an event, whereas a trace tends to be continuous output of diagnostic data. |
| resource pool | Resource pools used for allocating resources in DWS. By binding a user to a resource pool, you can limit the priority of the jobs executed by the user and resources available to the jobs. |
| tenant | A database service user who runs services using allocated computing (CPU, memory, and I/O) and storage resources. Service level agreements (SLAs) are met through resource management and isolation. |
| minimum restoration point | A method used by DWS to ensure data consistency. During startup, DWS checks consistency between the latest WAL logs and the minimum restoration point. If the record location of the minimum restoration point is greater than that of the latest WAL logs, the database fails to start. |

21 SQL Syntax Reference

21.1 DWS SQL Overview

What Is SQL?

SQL is a standard computer language used to control the access to databases and manage data in databases.

SQL provides different statements to enable you to:

- Query data.
- Insert, update, and delete rows.
- Create, replace, modify, and delete objects.
- Control the access to a database and its objects.
- Maintain the consistency and integrity of a database.

SQL consists of commands and functions that are used to manage databases and database objects. SQL can also forcibly implement the rules for data types, expressions, and texts. Therefore, section "SQL Reference" describes data types, expressions, functions, and operators in addition to SQL syntax.

Development of SQL Standards

Released SQL standards are as follows:

- 1986: ANSI X3.135-1986, ISO/IEC 9075:1986, SQL-86
- 1989: ANSI X3.135-1989, ISO/IEC 9075:1989, SQL-89
- 1992: ANSI X3.135-1992, ISO/IEC 9075:1992, SQL-92 (SQL2)
- 1999: ISO/IEC 9075:1999, SQL:1999 (SQL3)
- 2003: ISO/IEC 9075:2003, SQL:2003 (SQL4)
- 2011: ISO/IEC 9075:200N, SQL:2011 (SQL5)

Supported SQL Standards

DWS is compatible with Postgres-XC features and supports the major features of SQL2, SQL3, and SQL4 by default, and some features of SQL5.

Languages Supported by DWS

DWS supports PL/pgSQL and PL/Java.

SQL Example Description

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

21.2 DWS SQL Syntax

To better understand how to use the syntax, you can refer to the following description of SQL syntax text conventions.

Table 21-1 SQL Syntax Description

| Format | Description |
|--------------------------|--|
| Uppercase characters | Keywords must be in uppercase. |
| Lowercase characters | Parameters must be in lowercase. |
| [] | Optional syntax item. Items in brackets [] are optional. |
| { } | Mandatory syntax item. |
| ... | Preceding elements can appear repeatedly. |
| [x y ...] | One item is selected from two or more options or no item is selected. |
| { x y ... } | One item is selected from two or more options. |
| [x y ...] [...] | You can choose either multiple parameters or no parameters. If you choose multiple parameters, simply separate them with spaces. |
| [x y ...] [,...] | You can choose either multiple parameters or no parameters. If you choose multiple parameters, simply separate them with commas (,). |
| { x y ... } [...] | You must select at least one parameter. If you select multiple parameters, separate them with spaces. |
| { x y ... } [,...] | You must select at least one parameter. If you select multiple parameters, separate them with commas (,). |

21.3 Differences Between DWS and PostgreSQL

The differences between DWS and PostgreSQL are sorted based on PostgreSQL 9.X. The differences are as follows:

Client Differences

DWS **gsql** differs from PostgreSQL **psql** in that the former has made the following changes to enhance security:

- User passwords cannot be set by running the **\password** meta-command.
- The **\i+**, **\ir+**, and **\include_relative+** meta-commands and the input and output parameter **-k** are added to encrypt imported and exported files.
- Historical command lines cannot be printed to files using the **\s** meta-command.
- SQL statements related to sensitive operations, such as those containing passwords, are not recorded. Users cannot see such records when they turn pages or press up or down arrow keys to view the SQL history.
- After a connection is set up, a message is displayed to inform users of password expiration and to show version information.

gsql provides the following additional functions based on **psql**:

- The output format parameter **-r** is added to allow you to adjust the focus by pressing the **Tab** key or arrow keys when entering commands.
- The **\parallel** meta-command is added to improve execution performance.
- The **\set RETRY** meta-command is added to support retry upon statement errors.
- Slashes (/) are used as the default terminator at the end of PL/SQL statements **CREATE OR REPLACE FUNCTION/PROCEDURE**.

libpq:

During the development of certain DWS functions such as the gsql client connection tool, PostgreSQL libpq is greatly modified. However, the libpq interface is not verified in application development. Therefore, underlying risks probably exist. You can use the ODBC or JDBC APIs instead.

SQL Statement Differences

Table 21-2 PostgreSQL syntaxes not supported by DWS

| Type | Syntaxes Not Supported by DWS | Description |
|------|-------------------------------|---|
| Data | Geometric lines | For details about the data types supported by DWS, see Data Types . |
| | pg_node_tree | |

| Type | Syntaxes Not Supported by DWS | Description |
|------------|--|--|
| Functions | Enumeration functions: <ul style="list-style-type: none">enum_first(anyenum)enum_last(anyenum)enum_range(anyenum)enum_range(anyenum, anyenum) | For details about the functions supported by DWS, see Functions and Operators . |
| | Access permission query functions: <ul style="list-style-type: none">has_sequence_privilege(user, sequence, privilege)has_sequence_privilege(sequence, privilege) | |
| | System directory information functions: <ul style="list-style-type: none">pg_get_triggerdef(trigger_oid)pg_get_triggerdef(trigger_oid, pretty_bool) | |
| | Geometric type conversion function:
line(point, point) | |
| | pg_node_tree | |
| SQL syntax | CREATE TABLE clause:
INHERITS (parent_table [, ...]) | Inherits tables. |
| | CREATE TABLE column constraints:
REFERENCES reftable [(refcolumn)]
[MATCH FULL MATCH PARTIAL MATCH SIMPLE] [ON DELETE action] [ON UPDATE action] | Use REFERENCES reftable [(refcolumn)] [MATCH FULL MATCH PARTIAL MATCH SIMPLE] [ON DELETE action] [ON UPDATE action] to create a foreign key constraint for a table. |
| | CREATE TABLE table constraints:
EXCLUDE [USING index_method]
(exclude_element WITH operator [, ...]) | Use EXCLUDE [USING index_method] (exclude_element WITH operator [, ...]) to create exclusion constraints for a table. |
| | CREATE/ALTER/DROP EXTENSION | Loads, modifies, and deletes extensions. |
| | CREATE/ALTER/DROP AGGREGATE | Defines, modifies, and deletes aggregate functions. |

| Type | Syntaxes Not Supported by DWS | Description |
|------|--|---|
| | CREATE/ALTER/DROP OPERATOR | Creates, modifies, and deletes operators. |
| | CREATE/ALTER/DROP OPERATOR CLASS | Creates, modifies, and deletes operator classes. |
| | CREATE/ALTER/DROP OPERATOR FAMILY | Creates, modifies, and deletes operator families. |
| | CREATE/ALTER/DROP TEXT SEARCH PARSER | Creates, modifies, and deletes text search parsers. |
| | CREATE/ALTER/DROP TEXT SEARCH TEMPLATE | Create, modify, and delete text search templates. |
| | CREATE/ALTER/DROP COLLATION | Creates, modifies, and deletes collation rules. |
| | CREATE/ALTER/DROP CONVERSION | Defines, modifies, and deletes the conversion of character set encoding. |
| | CREATE/ALTER/DROP RULE | Creates, modifies, and deletes rules. |
| | CREATE/ALTER/DROP LANGUAGE | Registers, modifies, and deletes procedural languages (LANGUAGE). |
| | CREATE/ALTER/DROP DOMAIN | Creates, modifies, and deletes domains. |
| | CREATE/DROP CAST | Defines and deletes casts. |
| | CREATE/ALTER/DROP USER MAPPING | Defines, modifies, and deletes user mapping. |
| | SECURITY LABEL | Defines or changes the security tag of an object. |
| | NOTIFY | Generates a notification. |
| | LISTEN | Listens to a notification. |
| | UNLISTEN | Stops listening to a notification. |
| | LOAD | Loads or reloads a shared library file. |
| | DISCARD | Releases session resources of a database. (Clusters of 8.2.0 and later versions support DISCARD .) |

| Type | Syntaxes Not Supported by DWS | Description |
|-------|---------------------------------|---|
| | MOVE BACKWARD | Moves a cursor backward. |
| | COPY FROM FILE and COPY TO FILE | To isolate permissions, COPY FROM FILE and COPY TO FILE are disabled in DWS. |
| Other | User-defined C functions | For details about user-defined functions supported by DWS, see "Data Warehouse Service (DWS) Developer Guide" > "DWS User-Defined Functions". |

21.4 Keyword

The SQL contains reserved and non-reserved words. Reserved keywords must not be used as other identifiers in accordance with standards. Non-reserved keywords have special meanings only in specific environments and can be used as identifiers in other environments. Even so, do not use non-reserved keywords.

You can use **SELECT * FROM pg_get_keywords();** to query the keyword of DWS.

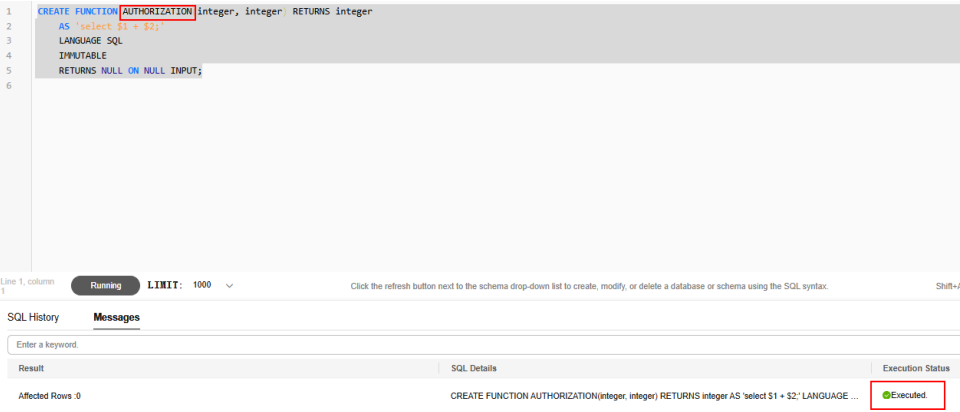
Example: **AUTHORIZATION** is a reserved keyword. Using it as a table name causes an error and prevents the table from being created.

However, you can use **AUTHORIZATION** as a function or type name under certain conditions (see the table below).

Figure 21-1 Using a keyword as a table name



Figure 21-2 Using a keyword as a function name



Identifier Naming Conventions

A database identifier is a name that uniquely identifies an object (such as a table, column, or index) in a database. In a database, identifiers are often used to reference database objects in SQL query statements to ensure the accuracy and consistency of database operations. It is important to name and use identifiers properly. The naming of DWS identifiers must comply with the following rules:

- The identifier must consist of letters (a-z), underscores (_), digits (0-9), and dollar signs (\$).
- An identifier name should start with a letter (a-z) or an underscore (_).
- The identifier can contain a maximum of 63 characters.
- Avoid using SQL reserved words as identifiers. Be careful with non-reserved keywords too.
- If the database uses GBK, UTF8, or SQL_ASCII, identifiers can include Chinese characters. In UTF8 or SQL_ASCII, each Chinese character takes up three bytes, with a maximum of 21 characters. In GBK, each Chinese character takes up two bytes, with a maximum of 31 characters. The **Latin1** character set does not support Chinese characters. The character set is set when creating the database. For details, see [CREATE DATABASE](#).

Table 21-3 SQL keyword list

| Keyword | DWS | SQL:1999 | SQL:1992 |
|----------|--------------|--------------|----------|
| ABORT | Non-reserved | - | - |
| ABS | - | Non-reserved | - |
| ABSOLUTE | Non-reserved | Reserved | Reserved |
| ACCESS | Non-reserved | - | - |
| ACCOUNT | Non-reserved | - | - |
| ACTION | Non-reserved | Reserved | Reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|------------|--------------|--------------|--------------|
| ADA | - | Non-reserved | Non-reserved |
| ADD | Non-reserved | Reserved | Reserved |
| ADMIN | Non-reserved | Reserved | - |
| AFTER | Non-reserved | Reserved | - |
| AGGREGATE | Non-reserved | Reserved | - |
| ALIAS | - | Reserved | - |
| ALL | Reserved | Reserved | Reserved |
| ALLOCATE | - | Reserved | Reserved |
| ALSO | Non-reserved | - | - |
| ALTER | Non-reserved | Reserved | Reserved |
| ALWAYS | Non-reserved | - | - |
| ANALYSE | Reserved | - | - |
| ANALYZE | Reserved | - | - |
| AND | Reserved | Reserved | Reserved |
| ANY | Reserved | Reserved | Reserved |
| APP | Non-reserved | - | - |
| ARE | - | Reserved | Reserved |
| ARRAY | Reserved | Reserved | - |
| AS | Reserved | Reserved | Reserved |
| ASC | Reserved | Reserved | Reserved |
| ASENSITIVE | - | Non-reserved | - |
| ASSERTION | Non-reserved | Reserved | Reserved |
| ASSIGNMENT | Non-reserved | Non-reserved | - |
| ASYMMETRIC | Reserved | Non-reserved | - |
| AT | Non-reserved | Reserved | Reserved |
| ATOMIC | - | Non-reserved | - |
| ATTRIBUTE | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|----------------|--|--------------|----------|
| AUTHID | Reserved | - | - |
| AUTHINFO | Non-reserved | - | - |
| AUTHORIZATION | Reserved (functions and types allowed) | Reserved | Reserved |
| AUTO | Non-reserved | - | - |
| AUTOEXTEND | Non-reserved | - | - |
| AUTOMAPPED | Non-reserved | - | - |
| AUTOREFRESH | Non-reserved | - | - |
| AUTOSCALE | Non-reserved | - | - |
| AVG | - | Non-reserved | Reserved |
| BACKWARD | Non-reserved | - | - |
| BARRIER | Non-reserved | - | - |
| BEFORE | Non-reserved | Reserved | - |
| BEGIN | Non-reserved | Reserved | Reserved |
| BETWEEN | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| BIGINT | Non-reserved (excluding functions and types) | - | - |
| BINARY | Reserved (functions and types allowed) | Reserved | - |
| BINARY_DOUBLE | Non-reserved (excluding functions and types) | - | - |
| BINARY_INTEGER | Non-reserved (excluding functions and types) | - | - |
| BIT | Non-reserved (excluding functions and types) | Reserved | Reserved |
| BITVAR | - | Non-reserved | - |
| BIT_LENGTH | - | Non-reserved | Reserved |
| BLOB | Non-reserved | Reserved | - |
| BLOCK | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------------------|--|--------------|--------------|
| BOOLEAN | Non-reserved (excluding functions and types) | Reserved | - |
| BOTH | Reserved | Reserved | Reserved |
| BUCKETS | Reserved | - | - |
| BUILD | Non-reserved | - | - |
| BREADTH | - | Reserved | - |
| BY | Non-reserved | Reserved | Reserved |
| C | - | Non-reserved | Non-reserved |
| CACHE | Non-reserved | - | - |
| CALL | Non-reserved | Reserved | - |
| CALLED | Non-reserved | Non-reserved | - |
| CARDINALITY | - | Non-reserved | - |
| CASCADE | Non-reserved | Reserved | Reserved |
| CASCADEDED | Non-reserved | Reserved | Reserved |
| CASE | Reserved | Reserved | Reserved |
| CAST | Reserved | Reserved | Reserved |
| CATALOG | Non-reserved | Reserved | Reserved |
| CATALOG_NAME | - | Non-reserved | Non-reserved |
| CHAIN | Non-reserved | Non-reserved | - |
| CHANGE | Non-reserved | - | - |
| CHAR | Non-reserved (excluding functions and types) | Reserved | Reserved |
| CHARACTER | Non-reserved (excluding functions and types) | Reserved | Reserved |
| CHARACTERISTICS | Non-reserved | - | - |
| CHARACTER_LENGTH | - | Non-reserved | Reserved |
| CHARACTER_SET_CATALOG | - | Non-reserved | Non-reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|----------------------|--|--------------|--------------|
| CHARACTER_SET_NAME | - | Non-reserved | Non-reserved |
| CHARACTER_SET_SCHEMA | - | Non-reserved | Non-reserved |
| CHAR_LENGTH | - | Non-reserved | Reserved |
| CHECK | Reserved | Reserved | Reserved |
| CHECKED | - | Non-reserved | - |
| CHECKPOINT | Non-reserved | - | - |
| CLASS | Non-reserved | Reserved | - |
| CLEAN | Non-reserved | - | - |
| CLASS_ORIGIN | - | Non-reserved | Non-reserved |
| CLOB | Non-reserved | Reserved | - |
| CLOSE | Non-reserved | Reserved | Reserved |
| CLUSTER | Non-reserved | - | - |
| COALESCE | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| COBOL | - | Non-reserved | Non-reserved |
| COLLATE | Reserved | Reserved | Reserved |
| COLLATION | Reserved (functions and types allowed) | Reserved | Reserved |
| COLLATION_CATALOG | - | Non-reserved | Non-reserved |
| COLLATION_NAME | - | Non-reserved | Non-reserved |
| COLLATION_SCHEMA | - | Non-reserved | Non-reserved |
| COLUMN | Reserved | Reserved | Reserved |
| COLUMNS | Non-reserved | - | - |
| COLUMN_NAME | - | Non-reserved | Non-reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|--------------------------|--|--------------|--------------|
| COMMAND_FUNCTION | - | Non-reserved | Non-reserved |
| COMMAND_FUNCTION_CODE | - | Non-reserved | - |
| COMMENT | Non-reserved | - | - |
| COMMENTS | Non-reserved | - | - |
| COMMIT | Non-reserved | Reserved | Reserved |
| COMMITTED | Non-reserved | Non-reserved | Non-reserved |
| COMPACT | Reserved (functions and types allowed) | - | - |
| COMPATIBLE_ILLEGAL_CHARS | Non-reserved | - | - |
| COMPLETE | Non-reserved | - | - |
| COMPRESS | Non-reserved | - | - |
| COMPLETION | - | Reserved | - |
| CONCURRENTLY | Reserved (functions and types allowed) | - | - |
| CONDITION | - | - | - |
| CONDITION_NUMBER | - | Non-reserved | Non-reserved |
| CONFIGURATION | Non-reserved | - | - |
| CONFLICT | Non-reserved | - | - |
| CONNECT | - | Reserved | Reserved |
| CONNECTION | Non-reserved | Reserved | Reserved |
| CONNECTION_NAME | - | Non-reserved | Non-reserved |
| CONSTRAINT | Reserved | Reserved | Reserved |
| CONSTRAINTS | Non-reserved | Reserved | Reserved |
| CONSTRAINT_CATALOG | - | Non-reserved | Non-reserved |
| CONSTRAINT_NAME | - | Non-reserved | Non-reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-------------------|--|--------------|--------------|
| CONSTRAINT_SCHEMA | - | Non-reserved | Non-reserved |
| CONSTRUCTOR | - | Reserved | - |
| CONTAINS | - | Non-reserved | - |
| CONTENT | Non-reserved | - | - |
| CONTINUE | Non-reserved | Reserved | Reserved |
| CONVERSION | Non-reserved | - | - |
| CONVERT | - | Non-reserved | Reserved |
| COORDINATOR | Non-reserved | - | - |
| COPY | Non-reserved | - | - |
| CORRESPONDING | - | Reserved | Reserved |
| COST | Non-reserved | - | - |
| COUNT | - | Non-reserved | Reserved |
| CREATE | Reserved | Reserved | Reserved |
| CROSS | Reserved (functions and types allowed) | Reserved | Reserved |
| CSV | Non-reserved | - | - |
| CUBE | Non-reserved | Reserved | - |
| CURRENT | Non-reserved | Reserved | Reserved |
| CURRENT_CATALOG | Reserved | - | - |
| CURRENT_DATE | Reserved | Reserved | Reserved |
| CURRENT_PATH | - | Reserved | - |
| CURRENT_ROLE | Reserved | Reserved | - |
| CURRENT_SCHEMA | Reserved (functions and types allowed) | - | - |
| CURRENT_TIME | Reserved | Reserved | Reserved |
| CURRENT_TIMESTAMP | Reserved | Reserved | Reserved |
| CURRENT_USER | Reserved | Reserved | Reserved |
| CURSOR | Non-reserved | Reserved | Reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------------------------|--|--------------|--------------|
| CURSOR_NAME | - | Non-reserved | Non-reserved |
| CYCLE | Non-reserved | Reserved | - |
| DATA | Non-reserved | Reserved | Non-reserved |
| DATE_FORMAT | Non-reserved | - | - |
| DATABASE | Non-reserved | - | - |
| DATAFILE | Non-reserved | - | - |
| DATETIME_INTERVAL_CODE | - | Non-reserved | Non-reserved |
| DATETIME_INTERVAL_PRECISION | - | Non-reserved | Non-reserved |
| DAY | Non-reserved | Reserved | Reserved |
| DBCOMPATIBILITY | Non-reserved | - | - |
| DBCOMPATIBILITY_BEHAVIOR | Non-reserved | - | - |
| DEALLOCATE | Non-reserved | Reserved | Reserved |
| DEC | Non-reserved (excluding functions and types) | Reserved | Reserved |
| DECIMAL | Non-reserved (excluding functions and types) | Reserved | Reserved |
| DECLARE | Non-reserved | Reserved | Reserved |
| DECODE | Non-reserved (excluding functions and types) | - | - |
| DEFAULT | Reserved | Reserved | Reserved |
| DEFAULTS | Non-reserved | - | - |
| DEFERRABLE | Reserved | Reserved | Reserved |
| DEFERRED | Non-reserved | Reserved | Reserved |
| DEFINED | - | Non-reserved | - |
| DEFINER | Non-reserved | Non-reserved | - |
| DELETE | Non-reserved | Reserved | Reserved |
| DELIMITER | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|---------------|--|--------------|----------|
| DELIMITERS | Non-reserved | - | - |
| DELTA | Non-reserved | - | - |
| DELTAMERGE | Reserved (functions and types allowed) | - | - |
| DEMAND | Non-reserved | - | - |
| DEPTH | - | Reserved | - |
| DEREF | - | Reserved | - |
| DESC | Reserved | Reserved | Reserved |
| DESCRIBE | - | Reserved | Reserved |
| DESCRIPTOR | - | Reserved | Reserved |
| DESTROY | - | Reserved | - |
| DESTRUCTOR | - | Reserved | - |
| DETERMINISTIC | Non-reserved | Reserved | - |
| DIAGNOSTICS | - | Reserved | Reserved |
| DICTIONARY | Non-reserved | Reserved | - |
| DIRECT | Non-reserved | - | - |
| DIRECTORY | Non-reserved | - | - |
| DISABLE | Non-reserved | - | - |
| DISCARD | Non-reserved | - | - |
| DISCONNECT | - | Reserved | Reserved |
| DISPATCH | - | Non-reserved | - |
| DISTINCT | Reserved | Reserved | Reserved |
| DISTRIBUTE | Non-reserved | - | - |
| DISTRIBUTION | Non-reserved | - | - |
| DO | Reserved | - | - |
| DOCUMENT | Non-reserved | - | - |
| DOMAIN | Non-reserved | Reserved | Reserved |
| DOUBLE | Non-reserved | Reserved | Reserved |
| DROP | Non-reserved | Reserved | Reserved |
| DROPCOLUMNS | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------------------|--------------|--------------|--------------|
| DUPLICATE | Non-reserved | - | - |
| DYNAMIC | - | Reserved | - |
| DYNAMIC_FUNCTION | - | Non-reserved | Non-reserved |
| DYNAMIC_FUNCTION_CODE | - | Non-reserved | - |
| EACH | Non-reserved | Reserved | - |
| ELASTIC | Non-reserved | - | - |
| ELSE | Reserved | Reserved | Reserved |
| ENABLE | Non-reserved | - | - |
| ENCODING | Non-reserved | - | - |
| ENCRYPTED | Non-reserved | - | - |
| END | Reserved | Reserved | Reserved |
| END-EXEC | - | Reserved | Reserved |
| ENFORCED | Non-reserved | - | - |
| ENUM | Non-reserved | - | - |
| EOL | Non-reserved | - | - |
| EQUALS | - | Reserved | - |
| ERRORS | Non-reserved | - | - |
| ESCAPE | Non-reserved | Reserved | Reserved |
| ESCAPING | Non-reserved | - | - |
| EVERY | Non-reserved | Reserved | - |
| EXCEPT | Reserved | Reserved | Reserved |
| EXCEPTION | - | Reserved | Reserved |
| EXCHANGE | Non-reserved | - | - |
| EXCLUDE | Non-reserved | - | - |
| EXCLUDING | Non-reserved | - | - |
| EXCLUSIVE | Non-reserved | - | - |
| EXEC | - | Reserved | Reserved |
| EXECUTE | Non-reserved | Reserved | Reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|---------------------|--|--------------|----------|
| EXISTING | - | Non-reserved | - |
| EXISTS | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| EXPIRATION | Non-reserved | - | - |
| EXPLAIN | Non-reserved | - | - |
| EXTENSION | Non-reserved | - | - |
| EXTERNAL | Non-reserved | Reserved | Reserved |
| EXTRACT | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| FALSE | Reserved | Reserved | Reserved |
| FAMILY | Non-reserved | - | - |
| FAST | Non-reserved | - | - |
| FENCED | Reserved | - | - |
| FETCH | Reserved | Reserved | Reserved |
| FILEHEADER | Non-reserved | - | - |
| FILL_AVG | Non-reserved (excluding functions and types) | - | - |
| FILL_FIRST | Non-reserved (excluding functions and types) | - | - |
| FILL_LAST | Non-reserved (excluding functions and types) | - | - |
| FILL_MISSING_FIELDS | Non-reserved | - | - |
| FILTER | Non-reserved | - | - |
| FINAL | - | Non-reserved | - |
| FIRST | Non-reserved | Reserved | Reserved |
| FIXED | Non-reserved | Reserved | Reserved |
| FLOAT | Non-reserved (excluding functions and types) | Reserved | Reserved |
| FOLLOWING | Non-reserved | - | - |
| FOR | Reserved | Reserved | Reserved |
| FORCE | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------|--|--------------|--------------|
| FOREIGN | Reserved | Reserved | Reserved |
| FORMATTER | Non-reserved | - | - |
| FORTRAN | - | Non-reserved | Non-reserved |
| FORWARD | Non-reserved | - | - |
| FOUND | - | Reserved | Reserved |
| FREE | - | Reserved | - |
| FREEZE | Reserved (functions and types allowed) | - | - |
| FROM | Reserved | Reserved | Reserved |
| FULL | Reserved (functions and types allowed) | Reserved | Reserved |
| FUNCTION | Non-reserved | Reserved | - |
| FUNCTIONS | Non-reserved | - | - |
| G | - | Non-reserved | - |
| GENERAL | - | Reserved | - |
| GENERATED | - | Non-reserved | - |
| GET | - | Reserved | Reserved |
| GLOBAL | Non-reserved | Reserved | Reserved |
| GO | - | Reserved | Reserved |
| GOTO | - | Reserved | Reserved |
| GRANT | Reserved | Reserved | Reserved |
| GRANTED | Non-reserved | Non-reserved | - |
| GREATEST | Non-reserved (excluding functions and types) | - | - |
| GROUP | Reserved | Reserved | Reserved |
| GROUPING | Non-reserved (excluding functions and types) | Reserved | - |
| HANDLER | Non-reserved | - | - |
| HAVING | Reserved | Reserved | Reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-------------------|--|--------------|----------|
| HDFS DIRECTORY | Reserved (functions and types allowed) | - | - |
| HEADER | Non-reserved | - | - |
| HIERARCHY | - | Non-reserved | - |
| HOLD | Non-reserved | Non-reserved | - |
| HOST | - | Reserved | - |
| HOT | Reserved | - | - |
| HOURL | Non-reserved | Reserved | Reserved |
| IDENTIFIED | Non-reserved | - | - |
| IDENTITY | Non-reserved | Reserved | Reserved |
| IF | Non-reserved (excluding functions and types) | - | - |
| IFNULL | Non-reserved (excluding functions and types) | - | - |
| IGNORE | Non-reserved | Reserved | - |
| IGNORE_EXTRA_DATA | Non-reserved | - | - |
| ILIKE | Reserved (functions and types allowed) | - | - |
| IMMEDIATE | Non-reserved | Reserved | Reserved |
| IMMUTABLE | Non-reserved | - | - |
| IMPLEMENTATION | - | Non-reserved | - |
| IMPLICIT | Non-reserved | - | - |
| IN | Reserved | Reserved | Reserved |
| INCLUDING | Non-reserved | - | - |
| INCREMENT | Non-reserved | - | - |
| INDEX | Non-reserved | - | - |
| INDEXES | Non-reserved | - | - |
| INDICATOR | - | Reserved | Reserved |
| INFIX | - | Non-reserved | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|--------------|--|--------------|----------|
| INHERIT | Non-reserved | - | - |
| INHERITS | Non-reserved | - | - |
| INITIAL | Non-reserved | - | - |
| INITIALIZE | - | Reserved | - |
| INITIALLY | Reserved | Reserved | Reserved |
| INITTRANS | Non-reserved | - | - |
| INLINE | Non-reserved | - | - |
| INNER | Reserved (functions and types allowed) | Reserved | Reserved |
| INOUT | Non-reserved (excluding functions and types) | Reserved | - |
| INPUT | Non-reserved | Reserved | Reserved |
| INSENSITIVE | Non-reserved | Non-reserved | Reserved |
| INSERT | Non-reserved | Reserved | Reserved |
| INSTANCE | - | Non-reserved | - |
| INSTANTIABLE | - | Non-reserved | - |
| INSTEAD | Non-reserved | - | - |
| INT | Non-reserved (excluding functions and types) | Reserved | Reserved |
| INTEGER | Non-reserved (excluding functions and types) | Reserved | Reserved |
| INTERNAL | Reserved | - | - |
| INTERSECT | Reserved | Reserved | Reserved |
| INTERVAL | Non-reserved (excluding functions and types) | Reserved | Reserved |
| INTO | Reserved | Reserved | Reserved |
| INVALID | Non-reserved | - | - |
| INVOKER | Non-reserved | Non-reserved | - |
| IS | Reserved | Reserved | Reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|------------|--|--------------|--------------|
| ISNULL | Non-reserved (excluding functions and types) | - | - |
| ISOLATION | Non-reserved | Reserved | Reserved |
| ITERATE | - | Reserved | - |
| JOIN | Reserved (functions and types allowed) | Reserved | Reserved |
| K | - | Non-reserved | - |
| KEEP | Non-reserved | - | - |
| KEEPALIVE | Non-reserved | - | - |
| KEY | Non-reserved | Reserved | Reserved |
| KEY_MEMBER | - | Non-reserved | - |
| KEY_TYPE | - | Non-reserved | - |
| LABEL | Non-reserved | - | - |
| LANGUAGE | Non-reserved | Reserved | Reserved |
| LARGE | Non-reserved | Reserved | - |
| LAST | Non-reserved | Reserved | Reserved |
| LATERAL | - | Reserved | - |
| LC_COLLATE | Non-reserved | - | - |
| LC_CTYPE | Non-reserved | - | - |
| LEADING | Reserved | Reserved | Reserved |
| LEAKPROOF | Non-reserved | - | - |
| LEAST | Non-reserved (excluding functions and types) | - | - |
| LEFT | Reserved (functions and types allowed) | Reserved | Reserved |
| LENGTH | - | Non-reserved | Non-reserved |
| LESS | Reserved | Reserved | - |
| LEVEL | Non-reserved | Reserved | Reserved |
| LIGHT | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|----------------|--|--------------|----------|
| LIKE | Reserved (functions and types allowed) | Reserved | Reserved |
| LIMIT | Reserved | Reserved | - |
| LIST | Non-reserved | - | - |
| LISTEN | Non-reserved | - | - |
| LOAD | Non-reserved | - | - |
| LOCAL | Non-reserved | Reserved | Reserved |
| LOCALTIME | Reserved | Reserved | - |
| LOCALTIMESTAMP | Reserved | Reserved | - |
| LOCATION | Non-reserved | - | - |
| LOCATOR | - | Reserved | - |
| LOCK | Non-reserved | - | - |
| LOG | Non-reserved | - | - |
| LOGGING | Non-reserved | - | - |
| LOGIN | Non-reserved | - | - |
| LOOP | Non-reserved | - | - |
| LOWER | - | Non-reserved | Reserved |
| M | - | Non-reserved | - |
| MAP | - | Reserved | - |
| MAPPING | Non-reserved | - | - |
| MASKED | Non-reserved | - | - |
| MATCH | Non-reserved | Reserved | Reserved |
| MATCHED | Non-reserved | - | - |
| MATERIALIZED | Non-reserved | - | - |
| MAX | - | Non-reserved | Reserved |
| MAXEXTENTS | Non-reserved | - | - |
| MAXSIZE | Non-reserved | - | - |
| MAXTRANS | Non-reserved | - | - |
| MAXVALUE | Reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|----------------------|--|--------------|--------------|
| MERGE | Non-reserved | - | - |
| MESSAGE_LENGTH | - | Non-reserved | Non-reserved |
| MESSAGE_OCTET_LENGTH | - | Non-reserved | Non-reserved |
| MESSAGE_TEXT | - | Non-reserved | Non-reserved |
| METAADDRESS | Non-reserved | - | - |
| METHOD | - | Non-reserved | - |
| MIN | - | Non-reserved | Reserved |
| MINEXTENTS | Non-reserved | - | - |
| MINUS | Reserved | - | - |
| MINUTE | Non-reserved | Reserved | Reserved |
| MINVALUE | Non-reserved | - | - |
| MOD | - | Non-reserved | - |
| MODE | Non-reserved | - | - |
| MODIFIES | - | Reserved | - |
| MODIFY | Reserved | Reserved | - |
| MODULE | - | Reserved | Reserved |
| MONTH | Non-reserved | Reserved | Reserved |
| MORE | - | Non-reserved | Non-reserved |
| MOVE | Non-reserved | - | - |
| MOVEMENT | Non-reserved | - | - |
| MUMPS | - | Non-reserved | Non-reserved |
| NAME | Non-reserved | Non-reserved | Non-reserved |
| NAMES | Non-reserved | Reserved | Reserved |
| NATIONAL | Non-reserved (excluding functions and types) | Reserved | Reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|------------|--|--------------|--------------|
| NATURAL | Reserved (functions and types allowed) | Reserved | Reserved |
| NBUCKETS | Non-reserved | - | - |
| NCHAR | Non-reserved (excluding functions and types) | Reserved | Reserved |
| NCLOB | - | Reserved | - |
| NEW | - | Reserved | - |
| NEXT | Non-reserved | Reserved | Reserved |
| NLSSORT | Reserved | - | - |
| NO | Non-reserved | Reserved | Reserved |
| NOCOMPRESS | Non-reserved | - | - |
| NOCYCLE | Non-reserved | - | - |
| NODE | Non-reserved | - | - |
| NOLOGGING | Non-reserved | - | - |
| NOLOGIN | Non-reserved | - | - |
| NOMAXVALUE | Non-reserved | - | - |
| NOMINVALUE | Non-reserved | - | - |
| NONE | Non-reserved (excluding functions and types) | Reserved | - |
| NOT | Reserved | Reserved | Reserved |
| NOTHING | Non-reserved | - | - |
| NOTIFY | Non-reserved | - | - |
| NOTNULL | Reserved (functions and types allowed) | - | - |
| NOWAIT | Non-reserved | - | - |
| NULL | Reserved | Reserved | Reserved |
| NULLABLE | - | Non-reserved | Non-reserved |
| NULLIF | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| NULLS | Non-reserved | - | - |
| NUMBER | Non-reserved (excluding functions and types) | Non-reserved | Non-reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|--------------|--|--------------|----------|
| NUMERIC | Non-reserved (excluding functions and types) | Reserved | Reserved |
| NUMSTR | Non-reserved | - | - |
| NVARCHAR2 | Non-reserved (excluding functions and types) | - | - |
| NVL | Non-reserved (excluding functions and types) | - | - |
| OBJECT | Non-reserved | Reserved | - |
| OCTET_LENGTH | - | Non-reserved | Reserved |
| OF | Non-reserved | Reserved | Reserved |
| OFF | Non-reserved | Reserved | - |
| OFFSET | Reserved | - | - |
| OIDS | Non-reserved | - | - |
| OLD | - | Reserved | - |
| ON | Reserved | Reserved | Reserved |
| ONLY | Reserved | Reserved | Reserved |
| OPEN | - | Reserved | Reserved |
| OPERATION | - | Reserved | - |
| OPERATOR | Non-reserved | - | - |
| OPTIMIZATION | Non-reserved | - | - |
| OPTION | Non-reserved | Reserved | Reserved |
| OPTIONS | Non-reserved | Non-reserved | - |
| OR | Reserved | Reserved | Reserved |
| ORDER | Reserved | Reserved | Reserved |
| ORDINALITY | Non-reserved | Reserved | - |
| OUT | Non-reserved (excluding functions and types) | Reserved | - |
| OUTER | Reserved (functions and types allowed) | Reserved | Reserved |
| OUTLINE | Non-reserved | - | - |
| OUTPUT | - | Reserved | Reserved |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|----------------------------|--|--------------|--------------|
| OVER | Non-reserved | - | - |
| OVERLAPS | Reserved (functions and types allowed) | Non-reserved | Reserved |
| OVERLAY | Non-reserved (excluding functions and types) | Non-reserved | - |
| OVERRIDING | - | Non-reserved | - |
| OVERWRITE | Non-reserved | - | - |
| OWNED | Non-reserved | - | - |
| OWNER | Non-reserved | - | - |
| PACKAGE | Non-reserved | - | - |
| PAD | - | Reserved | Reserved |
| PARAMETER | - | Reserved | - |
| PARAMETERS | - | Reserved | - |
| PARAMETER_MODE | - | Non-reserved | - |
| PARAMETER_NAME | - | Non-reserved | - |
| PARAMETER_ORDINAL_POSITION | - | Non-reserved | - |
| PARAMETER_SPECIFIC_CATALOG | - | Non-reserved | - |
| PARAMETER_SPECIFIC_NAME | - | Non-reserved | - |
| PARAMETER_SPECIFIC_SCHEMA | - | Non-reserved | - |
| PARENT_NAME | Non-reserved | - | - |
| PARSER | Non-reserved | - | - |
| PARTIAL | Non-reserved | Reserved | Reserved |
| PARTITION | Non-reserved | - | - |
| PARTITIONS | Non-reserved | - | - |
| PASCAL | - | Non-reserved | Non-reserved |
| PASSING | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------------|--|--------------|--------------|
| PASSWORD | Non-reserved | - | - |
| PATH | - | Reserved | - |
| PCTFREE | Non-reserved | - | - |
| PER | Non-reserved | - | - |
| PERM | Non-reserved | - | - |
| PERCENT | Non-reserved | - | - |
| PERFORMANCE | Reserved | - | - |
| PLACING | Reserved | - | - |
| PLAN | Reserved | - | - |
| PLANS | Non-reserved | - | - |
| PLI | - | Non-reserved | Non-reserved |
| POLICY | Non-reserved | - | - |
| POOL | Non-reserved | - | - |
| POSITION | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| POSTFIX | - | Reserved | - |
| PRECEDING | Non-reserved | - | - |
| PRECISION | Non-reserved (excluding functions and types) | Reserved | Reserved |
| PREDICATE | Non-reserved | - | - |
| PREFERRED | Non-reserved | - | - |
| PREFIX | Non-reserved | Reserved | - |
| PREORDER | - | Reserved | - |
| PREPARE | Non-reserved | Reserved | Reserved |
| PREPARED | Non-reserved | - | - |
| PRESERVE | Non-reserved | Reserved | Reserved |
| PRESERVE_BLANKS | Non-reserved | - | - |
| PRIMARY | Reserved | Reserved | Reserved |
| PRIOR | Non-reserved | Reserved | Reserved |
| PRIVATE | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-------------|--|----------|----------|
| PRIVILEGE | Non-reserved | - | - |
| PRIVILEGES | Non-reserved | Reserved | Reserved |
| PROCEDURAL | Non-reserved | - | - |
| PROCEDURE | Reserved | Reserved | Reserved |
| PROFILE | Non-reserved | - | - |
| PUBLIC | - | Reserved | Reserved |
| PUBLICATION | Non-reserved | - | - |
| PURGE | Non-reserved | - | - |
| QUERY | Non-reserved | - | - |
| QUOTE | Non-reserved | - | - |
| RANGE | Non-reserved | - | - |
| RAW | Non-reserved | - | - |
| READ | Non-reserved | Reserved | Reserved |
| READS | - | Reserved | - |
| REAL | Non-reserved (excluding functions and types) | Reserved | Reserved |
| REASSIGN | Non-reserved | - | - |
| REBUILD | Non-reserved | - | - |
| RECHECK | Non-reserved | - | - |
| RECURSIVE | Non-reserved | Reserved | - |
| RECYCLEBIN | Reserved (functions and types allowed) | - | - |
| REDACTION | Non-reserved | - | - |
| REF | Non-reserved | Reserved | - |
| REFRESH | Non-reserved | - | - |
| REFERENCES | Reserved | Reserved | Reserved |
| REFERENCING | - | Reserved | - |
| REINDEX | Non-reserved | - | - |
| REJECT | Reserved | - | - |
| RELATIVE | Non-reserved | Reserved | Reserved |
| RELEASE | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------------------|--|--------------|--------------|
| REOPTIONS | Non-reserved | - | - |
| REMOTE | Non-reserved | - | - |
| RENAME | Non-reserved | - | - |
| REPEATABLE | Non-reserved | Non-reserved | Non-reserved |
| REPLACE | Non-reserved | - | - |
| REPLICA | Non-reserved | - | - |
| RESET | Non-reserved | - | - |
| RESIZE | Non-reserved | - | - |
| RESOURCE | Non-reserved | - | - |
| RESTART | Non-reserved | - | - |
| RESTRICT | Non-reserved | Reserved | Reserved |
| RESULT | - | Reserved | - |
| RETURN | Non-reserved | Reserved | - |
| RETURNED_LENGTH | - | Non-reserved | Non-reserved |
| RETURNED_OCTET_LENGTH | - | Non-reserved | Non-reserved |
| RETURNED_SQLSTATE | - | Non-reserved | Non-reserved |
| RETURNING | Reserved | - | - |
| RETURNS | Non-reserved | Reserved | - |
| REUSE | Non-reserved | - | - |
| REVOKE | Non-reserved | Reserved | Reserved |
| REWRITE | Non-reserved | - | - |
| RIGHT | Reserved (functions and types allowed) | Reserved | Reserved |
| ROLE | Non-reserved | Reserved | - |
| ROLLBACK | Non-reserved | Reserved | Reserved |
| ROLLUP | Non-reserved | Reserved | - |
| ROUTINE | - | Reserved | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------------|--|--------------|--------------|
| ROUTINE_CATALOG | - | Non-reserved | - |
| ROUTINE_NAME | - | Non-reserved | - |
| ROUTINE_SCHEMA | - | Non-reserved | - |
| ROW | Non-reserved (excluding functions and types) | Reserved | - |
| ROWS | Non-reserved | Reserved | Reserved |
| ROW_COUNT | - | Non-reserved | Non-reserved |
| RULE | Non-reserved | - | - |
| SAVEPOINT | Non-reserved | Reserved | - |
| SCALE | - | Non-reserved | Non-reserved |
| SCHEMA | Non-reserved | Reserved | Reserved |
| SCHEMA_NAME | - | Non-reserved | Non-reserved |
| SCOPE | - | Reserved | - |
| SCROLL | Non-reserved | Reserved | Reserved |
| SEARCH | Non-reserved | Reserved | - |
| SECOND | Non-reserved | Reserved | Reserved |
| SECTION | - | Reserved | Reserved |
| SECURITY | Non-reserved | Non-reserved | - |
| SELECT | Reserved | Reserved | Reserved |
| SELF | - | Non-reserved | - |
| SENSITIVE | - | Non-reserved | - |
| SEPARATOR | Non-reserved | - | - |
| SEQUENCE | Non-reserved | Reserved | - |
| SEQUENCES | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|----------------------|--|--------------|--------------|
| SERIALIZABLE | Non-reserved | Non-reserved | Non-reserved |
| SERVER | Non-reserved | - | - |
| SERVER_NAME | - | Non-reserved | Non-reserved |
| SESSION | Non-reserved | Reserved | Reserved |
| SESSION_USER | Reserved | Reserved | Reserved |
| SET | Non-reserved | Reserved | Reserved |
| SETOF | Non-reserved (excluding functions and types) | - | - |
| SETS | Non-reserved | Reserved | - |
| SHARE | Non-reserved | - | - |
| SHIPPABLE | Non-reserved | - | - |
| SHOW | Non-reserved | - | - |
| SIMILAR | Reserved (functions and types allowed) | Non-reserved | - |
| SIMPLE | Non-reserved | Non-reserved | - |
| SIZE | Non-reserved | Reserved | Reserved |
| SLOT | Non-reserved | - | - |
| SMALLDATETIME | Non-reserved (excluding functions and types) | - | - |
| SMALLDATETIME_FORMAT | Non-reserved | - | - |
| SMALLINT | Non-reserved (excluding functions and types) | Reserved | Reserved |
| SNAPSHOT | Non-reserved | - | - |
| SOME | Reserved | Reserved | Reserved |
| SOURCE | Non-reserved | Non-reserved | - |
| SPACE | Non-reserved | Reserved | Reserved |
| SPECIFIC | - | Reserved | - |
| SPECIFICTYPE | - | Reserved | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-----------------|--------------|--------------|--------------|
| SPECIFIC_NAME | - | Non-reserved | - |
| SPILL | Non-reserved | - | - |
| SPLIT | Non-reserved | - | - |
| SQL | Non-reserved | Reserved | Reserved |
| SQLCODE | - | - | Reserved |
| SQLERROR | - | - | Reserved |
| SQLException | - | Reserved | - |
| SQLSTATE | - | Reserved | Reserved |
| SQLWARNING | - | Reserved | - |
| STABLE | Non-reserved | - | - |
| STANDALONE | Non-reserved | - | - |
| START | Non-reserved | Reserved | - |
| STATE | - | Reserved | - |
| STATEMENT | Non-reserved | Reserved | - |
| STATEMENT_ID | Non-reserved | - | - |
| STATIC | - | Reserved | - |
| STATISTICS | Non-reserved | - | - |
| STDIN | Non-reserved | - | - |
| STDOUT | Non-reserved | - | - |
| STORAGE | Non-reserved | - | - |
| STORE | Non-reserved | - | - |
| STRICT | Non-reserved | - | - |
| STRIP | Non-reserved | - | - |
| STRUCTURE | - | Reserved | - |
| STYLE | - | Non-reserved | - |
| SUBCLASS_ORIGIN | - | Non-reserved | Non-reserved |
| SUBLIST | - | Non-reserved | - |
| SUBSCRIPTION | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|---------------|--|--------------|--------------|
| SUBSTRING | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| SUM | - | Non-reserved | Reserved |
| SYMMETRIC | Reserved | Non-reserved | - |
| SYNONYM | Non-reserved | - | - |
| SYS_REFCURSOR | Non-reserved | - | - |
| SYSDATE | Reserved | - | - |
| SYSID | Non-reserved | - | - |
| SYSTEM | Non-reserved | Non-reserved | - |
| SYSTEM_USER | - | Reserved | Reserved |
| TABLE | Reserved | Reserved | Reserved |
| TABLES | Non-reserved | - | - |
| TABLESAMPLE | Reserved (functions and types allowed) | - | - |
| TABLE_NAME | - | Non-reserved | Non-reserved |
| TEMP | Non-reserved | - | - |
| TEMPLATE | Non-reserved | - | - |
| TEMPORARY | Non-reserved | Reserved | Reserved |
| TERMINATE | - | Reserved | - |
| TEXT | Non-reserved | - | - |
| THAN | Non-reserved | Reserved | - |
| THEN | Reserved | Reserved | Reserved |
| TIME | Non-reserved (excluding functions and types) | Reserved | Reserved |
| TIME_FILL | Non-reserved (excluding functions and types) | - | - |
| TIME_FORMAT | Non-reserved | - | - |
| TIMECAPSULE | Reserved (functions and types allowed) | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|--------------------------|--|--------------|----------|
| TIMESTAMP | Non-reserved (excluding functions and types) | Reserved | Reserved |
| TIMESTAMPADD | Non-reserved (excluding functions and types) | - | - |
| TIMESTAMPDIFF | Non-reserved (excluding functions and types) | - | - |
| TIMESTAMP_FORMAT | Non-reserved | - | - |
| TIMEZONE_HOUR | - | Reserved | Reserved |
| TIMEZONE_MINUTE | - | Reserved | Reserved |
| TINYINT | Non-reserved (excluding functions and types) | - | - |
| TO | Reserved | Reserved | Reserved |
| TRAILING | Reserved | Reserved | Reserved |
| TRANSACTION | Non-reserved | Reserved | Reserved |
| TRANSACTIONS_COMMITTED | - | Non-reserved | - |
| TRANSACTIONS_ROLLED_BACK | - | Non-reserved | - |
| TRANSACTION_ACTIVE | - | Non-reserved | - |
| TRANSFORM | - | Non-reserved | - |
| TRANSFORMS | - | Non-reserved | - |
| TRANSLATE | - | Non-reserved | Reserved |
| TRANSLATION | - | Reserved | Reserved |
| TREAT | Non-reserved (excluding functions and types) | Reserved | - |
| TRIGGER | Non-reserved | Reserved | - |
| TRIGGER_CATALOG | - | Non-reserved | - |
| TRIGGER_NAME | - | Non-reserved | - |
| TRIGGER_SCHEMA | - | Non-reserved | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|-------------|---|--------------|--------------|
| TRIM | Non-reserved (excluding functions and types) | Non-reserved | Reserved |
| TRUE | Reserved | Reserved | Reserved |
| TRUNCATE | Non-reserved | - | - |
| TRUSTED | Non-reserved | - | - |
| TRY_CAST | Non-reserved | - | - |
| TSTAG | Reserved. This field is used only in the hybrid data warehouse. | - | - |
| TSTIME | Reserved. This field is used only in the hybrid data warehouse. | - | - |
| TSFIELD | Reserved. This field is used only in the hybrid data warehouse. | - | - |
| TYPE | Non-reserved | Non-reserved | Non-reserved |
| TYPES | Non-reserved | - | - |
| UESCAPE | - | - | - |
| UNBOUNDED | Non-reserved | - | - |
| UNCOMMITTED | Non-reserved | Non-reserved | Non-reserved |
| UNDER | - | Reserved | - |
| UNENCRYPTED | Non-reserved | - | - |
| UNION | Reserved | Reserved | Reserved |
| UNIQUE | Reserved | Reserved | Reserved |
| UNKNOWN | Non-reserved | Reserved | Reserved |
| UNLIMITED | Non-reserved | - | - |
| UNLISTEN | Non-reserved | - | - |
| UNLOCK | Non-reserved | - | - |
| UNLOGGED | Non-reserved | - | - |
| UNNAMED | - | Non-reserved | Non-reserved |
| UNNEST | - | Reserved | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|---------------------------|--|--------------|----------|
| UNTIL | Non-reserved | - | - |
| UNUSABLE | Non-reserved | - | - |
| UPDATE | Non-reserved | Reserved | Reserved |
| UPPER | - | Non-reserved | Reserved |
| USAGE | - | Reserved | Reserved |
| USER | Reserved | Reserved | Reserved |
| USER_DEFINED_TYPE_CATALOG | - | Non-reserved | - |
| USER_DEFINED_TYPE_NAME | - | Non-reserved | - |
| USER_DEFINED_TYPE_SCHEMA | - | Non-reserved | - |
| USING | Reserved | Reserved | Reserved |
| VACUUM | Non-reserved | - | - |
| VALID | Non-reserved | - | - |
| VALIDATE | Non-reserved | - | - |
| VALIDATION | Non-reserved | - | - |
| VALIDATOR | Non-reserved | - | - |
| VALUE | Non-reserved | Reserved | Reserved |
| VALUES | Non-reserved (excluding functions and types) | Reserved | Reserved |
| VARCHAR | Non-reserved (excluding functions and types) | Reserved | Reserved |
| VARCHAR2 | Non-reserved (excluding functions and types) | - | - |
| VARIABLE | - | Reserved | - |
| VARIADIC | Reserved | - | - |
| VARYING | Non-reserved | Reserved | Reserved |
| VCGROUP | Non-reserved | - | - |
| VERBOSE | Reserved (functions and types allowed) | - | - |
| VERIFY | Non-reserved | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|---------------|--|----------|----------|
| VERSION | Non-reserved | - | - |
| VIEW | Non-reserved | Reserved | Reserved |
| VOLATILE | Non-reserved | - | - |
| WARMUP | Reserved | - | - |
| WHEN | Reserved | Reserved | Reserved |
| WHENEVER | - | Reserved | Reserved |
| WHERE | Reserved | Reserved | Reserved |
| WHITESPACE | Non-reserved | - | - |
| WINDOW | Reserved | - | - |
| WITH | Reserved | Reserved | Reserved |
| WITHIN | Non-reserved | - | - |
| WITHOUT | Non-reserved | Reserved | - |
| WORK | Non-reserved | Reserved | Reserved |
| WORKLOAD | Non-reserved | - | - |
| WRAPPER | Non-reserved | - | - |
| WRITE | Non-reserved | Reserved | Reserved |
| XML | Non-reserved | - | - |
| XMLATTRIBUTES | Non-reserved (excluding functions and types) | - | - |
| XMLCONCAT | Non-reserved (excluding functions and types) | - | - |
| XMLELEMENT | Non-reserved (excluding functions and types) | - | - |
| XMLEXISTS | Non-reserved (excluding functions and types) | - | - |
| XMLFOREST | Non-reserved (excluding functions and types) | - | - |
| XMLNAMESPACES | Non-reserved (excluding functions and types) | - | - |
| XMLPARSE | Non-reserved (excluding functions and types) | - | - |
| XMLPI | Non-reserved (excluding functions and types) | - | - |

| Keyword | DWS | SQL:1999 | SQL:1992 |
|--------------|--|----------|----------|
| XMLROOT | Non-reserved (excluding functions and types) | - | - |
| XMLSERIALIZE | Non-reserved (excluding functions and types) | - | - |
| XMLTABLE | Non-reserved (excluding functions and types) | - | - |
| YEAR | Non-reserved | Reserved | Reserved |
| YES | Non-reserved | - | - |
| ZONE | Non-reserved | Reserved | Reserved |

21.5 Data Types

21.5.1 Numeric Types

Numeric types consist of two-, four-, and eight-byte integers, four- and eight-byte floating-point numbers, and selectable-precision decimals.

For details about numeric operators and functions, see [Numeric Functions and Operators](#).

DWS supports integers, arbitrary precision numbers, floating point types, and serial integers.

Integer Types

The types TINYINT, SMALLINT, INTEGER, BINARY_INTEGER, and BIGINT store whole numbers, that is, numbers without fractional components, of various ranges. Saving a number with a decimal in any of the data types will result in errors.

The type INTEGER is the common choice. Generally, use the SMALLINT type only if you are sure that the value range is within the SMALLINT value range. The storage speed of INTEGER is much faster. BIGINT is used only when the range of INTEGER is not large enough.

Table 21-4 Integer types

| Column | Description | Storage Space | Range |
|---------|--------------------------------|---------------|---------|
| TINYINT | Tiny integer, also called INT1 | 1 byte | 0 ~ 255 |

| Column | Description | Storage Space | Range |
|----------------|--|---------------|--|
| SMALLINT | Small integer, also called INT2 | 2 bytes | -32,768 ~ +32,767 |
| INTEGER | Typical choice for integer, also called INT4 | 4 bytes | -2,147,483,648 ~ +2,147,483,647 |
| BINARY_INTEGER | INTEGER alias, compatible with Oracle | 4 bytes | -2,147,483,648 ~ +2,147,483,647 |
| BIGINT | Big integer, also called INT8 | 8 bytes | -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807 |

Examples:

Create a table containing TINYINT, INTEGER, and BIGINT data.

```
CREATE TABLE int_type_t1
(
  a TINYINT,
  b TINYINT,
  c INTEGER,
  d BIGINT
);
```

Insert data.

```
INSERT INTO int_type_t1 VALUES(100, 10, 1000, 10000);
```

View data.

```
SELECT * FROM int_type_t1;
 a | b | c | d
---+---+---+---
100 | 10 | 1000 | 10000
(1 row)
```

Arbitrary Precision Types

The type NUMBER can store numbers with a very large number of digits. It is especially recommended for storing monetary amounts and other quantities where exactness is required. The arbitrary precision numbers require larger storage space and have lower storage efficiency, operation efficiency, and poorer compression ratio results than integer types.

The scale of a NUMBER value is the count of decimal digits in the fractional part, to the right of the decimal point. The precision of a NUMBER value is the total count of significant digits in the whole number, that is, the number of digits to both sides of the decimal point. So the number 23.5141 has a precision of 6 and a scale of 4. Integers can be considered to have a scale of zero.

To configure a numeric or decimal column, you are advised to specify both the maximum precision (p) and the maximum scale (s) of the column.

If the precision or scale of a value is greater than the declared scale of the column, the system will round the value to the specified number of fractional digits. Then, if the number of digits to the left of the decimal point exceeds the declared precision minus the declared scale, an error will be reported.

Table 21-5 Any-precision types

| Column | Description | Storage Space | Range |
|---------------------------------------|---|--|---|
| NUMERIC[(p[,s])],
DECIMAL[(p[,s])] | The value range of p (precision) is [1,1000], and the value range of s (standard) is [0,p]. | The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data. | Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified |
| NUMBER[(p[,s])] | Alias for type NUMERIC, compatible with Oracle | The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data. | Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified |

Examples:

Create a table with DECIMAL values.

```
CREATE TABLE decimal_type_t1 (DT_COL1 DECIMAL(10,4));
```

Insert data.

```
INSERT INTO decimal_type_t1 VALUES(123456.122331);
```

View data.

```
SELECT * FROM decimal_type_t1;  
dt_col1  
-----  
123456.1223  
(1 row)
```

Floating-Point Types

The floating-point type is an inexact, variable-precision numeric type. This type is an implementation of IEEE Standard 754 for Binary Floating-Point Arithmetic (single and double precision, respectively), to the extent that the underlying processor, OS, and compiler support it.

Table 21-6 Floating point types

| Column | Description | Storage Space | Range |
|------------------------------------|--|--|---|
| REAL,
FLOAT4 | Single precision floating points, inexact | 4 bytes | Six bytes of decimal digits |
| DOUBLE
PRECISION
,
FLOAT8 | Double precision floating points, inexact | 8 bytes | 1E-307~1E+308,
15 bytes of decimal digits |
| FLOAT[(p)
] | Floating points, inexact. The value range of precision (p) is [1,53].
NOTE
p is the precision, indicating the total decimal digits. | 4 or 8 bytes | REAL or DOUBLE PRECISION is selected as an internal identifier based on different precision (p). If no precision is specified, DOUBLE PRECISION is used as the internal identifier. |
| BINARY_D
OUBLE | DOUBLE PRECISION alias, compatible with Oracle | 8 bytes | 1E-307~1E+308,
15 bytes of decimal digits |
| DEC[(p[,s])
] | The value range of p (precision) is [1,1000], and the value range of s (standard) is [0,p].
NOTE
p indicates the total digits, and s indicates the decimal digit. | The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data. | Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified |
| INTEGER[(
p[,s])] | The value range of p (precision) is [1,1000], and the value range of s (standard) is [0,p]. | The precision is specified by users. Every four decimal digits occupy two bytes, and an extra eight-byte overhead is added to the entire data. | Up to 131,072 digits before the decimal point; and up to 16,383 digits after the decimal point when no precision is specified |

Examples:

Create a table with floating-point values.

```
CREATE TABLE float_type_t2  
(
```

```
FT_COL1 INTEGER,  
FT_COL2 FLOAT4,  
FT_COL3 FLOAT8,  
FT_COL4 FLOAT(3),  
FT_COL5 BINARY_DOUBLE,  
FT_COL6 DECIMAL(10,4),  
FT_COL7 INTEGER(6,3)  
) DISTRIBUTE BY HASH ( ft_col1);
```

Insert data.

```
INSERT INTO float_type_t2 VALUES(10,10.365456,123456.1234,10.3214, 321.321, 123.123654, 123.123654);
```

View data.

```
SELECT * FROM float_type_t2;  
ft_col1 | ft_col2 | ft_col3 | ft_col4 | ft_col5 | ft_col6 | ft_col7  
-----+-----+-----+-----+-----+-----+-----  
10 | 10.3655 | 123456.1234 | 10.3214 | 321.321 | 123.1237 | 123.124  
(1 row)
```

Serial Integers

SMALLSERIAL, SERIAL, and BIGSERIAL are not true types, but merely a notational convenience for creating unique identifier columns. Therefore, an integer column is created and its default value plans to be read from a sequencer. A NOT NULL constraint is used to ensure NULL is not inserted. In most cases you would also want to attach a UNIQUE or PRIMARY KEY constraint to prevent duplicate values from being inserted unexpectedly. Lastly, the sequence is marked as "owned by" the column, so that it will be dropped if the column or table is dropped. Currently, the SERIAL column can be specified only when you create a table. You cannot add the SERIAL column in an existing table. In addition, SERIAL columns cannot be created in temporary tables. Because SERIAL is not a data type, columns cannot be converted to this type.

Table 21-7 Sequence integer

| Column | Description | Storage Space | Range |
|-------------|--------------------------------------|---------------|-------------------------------|
| SMALLSERIAL | Two-byte auto-incrementing integer | 2 bytes | 1 ~ 32,767 |
| SERIAL | Four-byte auto-incrementing integer | 4 bytes | 1 ~ 2,147,483,647 |
| BIGSERIAL | Eight-byte auto-incrementing integer | 8 bytes | 1 ~ 9,223,372,036,854,775,807 |

Examples:

Create a table with serial values.

```
CREATE TABLE smallserial_type_tab(a SMALLSERIAL);
```

Insert data.

```
INSERT INTO smallserial_type_tab VALUES(default);
```

Insert data again.

```
INSERT INTO smallserial_type_tab VALUES(default);
```

View data.

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

21.5.2 Monetary Types

The **money** type stores a currency amount with fixed fractional precision. The range shown in [Table 21-8](#) assumes there are two fractional digits. Input is accepted in a variety of formats, including integer and floating-point literals, as well as typical currency formatting, such as \$1,000.00. Output is generally in the latter form but depends on the locale.

Table 21-8 Monetary types

| Name | Storage Size | Description | Range |
|-------|--------------|-----------------|--|
| money | 8 bytes | Currency amount | -92233720368547758.08 to +92233720368547758.07 |

Values of the **numeric**, **int**, and **bigint** data types can be cast to **money**. Conversion from the **real** and **double precision** data types can be done by casting to **numeric** first, for example:

```
SELECT '12.34'::float8::numeric::money;
```

However, this is not recommended. Floating point numbers should not be used to handle money due to the potential for rounding errors.

A **money** value can be cast to **numeric** without loss of precision. Conversion to other types could potentially lose precision, and must also be done in two stages:

```
SELECT '52093.89'::money::numeric::float8;
```

When a **money** value is divided by another **money** value, the result is **double precision** (that is, a pure number, not money); the currency units cancel each other out in the division.

21.5.3 Boolean Type

Table 21-9 Boolean type

| Name | Description | Storage Space | Value |
|---------|--------------|---------------|--|
| BOOLEAN | Boolean type | 1 byte | <ul style="list-style-type: none">• true• false• null (unknown) |

Valid literal values for the "true" state are:

TRUE, 't', 'true', 'y', 'yes', '1'

Valid literal values for the "false" state include:

FALSE, 'f', 'false', 'n', 'no', '0'

TRUE and **FALSE** are standard expressions, compatible with SQL statements.

Examples

Data type **boolean** is displayed with letters **t** and **f**.

```
-- Create a table:
CREATE TABLE bool_type_t1
(
    BT_COL1 BOOLEAN,
    BT_COL2 TEXT
) DISTRIBUTE BY HASH(BT_COL2);

--Insert data:
INSERT INTO bool_type_t1 VALUES (TRUE, 'sic est');

INSERT INTO bool_type_t1 VALUES (FALSE, 'non est');

-- View data:
SELECT * FROM bool_type_t1;
bt_col1 | bt_col2
-----+-----
t      | sic est
f      | non est
(2 rows)

SELECT * FROM bool_type_t1 WHERE bt_col1 = 't';
bt_col1 | bt_col2
-----+-----
t      | sic est
(1 row)

-- Delete the tables:
DROP TABLE bool_type_t1;
```

21.5.4 Character Types

Table 21-10 lists the character types that can be used in DWS. For string operators and related built-in functions, see [String Processing Functions and Operators](#).

Table 21-10 Character types

| Name | Description | Length | Storage Space |
|---|--|--|---|
| CHAR (n) or
BPCHAR
CHARACTER(n)
NCHAR(n) | Fixed-length string. If the length exceeds the value of n , the excess part is truncated. If the length is less than the value of n , spaces are padded. | n indicates the byte length. If it is not specified, the default precision 1 is used. The value of n is less than 10485761 . | The maximum size is 10 MB. |
| VARCHAR(n)
CHARACTER
VARYING(n) | Variable-length string. | n indicates the byte length. The value of n is less than 10485761 . | The maximum size is 10 MB. |
| VARCHAR2(n) | Variable-length string. It is an alias for VARCHAR(n) type, compatible with Oracle. | n indicates the byte length. The value of n is less than 10485761 . | The maximum size is 10 MB. |
| NVARCHAR2(n) | Variable-length string. | n indicates the string length. The value of n is less than 10485761 . | The maximum size is 10 MB. |
| CLOB | Variable-length string. A big text object. It is an alias for TEXT type, compatible with Oracle. | - | The maximum size is 1,073,733,621 bytes (1 GB – 8,203 bytes). |
| TEXT | Variable-length string. | - | The maximum size is 1,073,733,621 bytes (1 GB – 8,203 bytes). |

 NOTE

- In addition to the size limitation on each column, the total size of each tuple is 1,073,733,621 bytes (1 GB – 8,023 bytes).
- For string data, you are advised to use variable-length strings and specify the maximum length. To avoid truncation, ensure that the specified maximum length is greater than the maximum number of characters to be stored. You are not advised to use fixed-length character types such as CHAR(n), NCHAR(n), and CHARACTER(n) unless you know that the data type is a fixed-length character string. In DWS, operations of the fixed-length character type will use extra storage and memory resources.
- Generally, if the data contains Chinese characters, use NCHAR/NVARCHAR2. If the data contains only English characters and digits, use CHAR/VARCHAR.

DWS has two other fixed-length character types, as listed in [Table 21-11](#).

The name type is used only in the internal system catalog as the storage identifier. The length of this type is 64 bytes (63 characters plus the terminator). This data type is not recommended for common users. When the name type is aligned with other data types (for example, in multiple branches of **case when**, one branch returns the name type and other branches return the text type), the name type may be aligned but characters may be truncated. If you do not want to have 64-bit truncated characters, you need to forcibly convert the name type to the text type.

The type **"char"** only uses one byte of storage. It is internally used in the system catalogs as a simplistic enumeration type.

Table 21-11 Special character types

| Name | Description | Storage Space |
|--------|--------------------------------|---------------|
| name | Internal type for object names | 64 bytes |
| "char" | Single-byte internal type | 1 byte |

Length

If a field is defined as **char(n)** or **varchar(n)**, **n** indicates the maximum length. No matter which type is used, the maximum length that can be set cannot exceed 10,485,760 bytes (that is, 10 MB).

When the data length exceeds the specified length **n**, the error "value too long" is reported. Of course, you can also specify to automatically truncate the data that exceeds the length.

Example:

1. Create table **t1** and specify the character type of its columns.

```
CREATE TABLE t1 (a char(5),b varchar(5));
```
2. An error is reported when the length of data inserted into the table **t1** exceeds the specified byte length.

```
INSERT INTO t1 VALUES('bookstore','123');  
ERROR: value too long for type character(5)  
CONTEXT: referenced column: a
```
3. Insert data into table **t1** and specify that the data is automatically truncated when the length exceeds the specified bytes.

```
INSERT INTO t1 VALUES('bookstore'::char(5),'12345678'::varchar(5));
INSERT 0 1

SELECT a,b FROM t1;
 a | b
-----+-----
books | 12345
(1 row)
```

Fixed Length and Variable Length

All character types can be classified into fixed-length strings and variable-length strings.

- For a fixed-length string, the length must be specified. If the length is not specified, the default length **1** is used. If the data length does not reach the specified length, spaces are automatically added to the end of the string. However, the added spaces are meaningless and will be ignored in actual operations, such as comparison, sorting, and type conversion.
- For a variable-length string, if the length is specified, the specified length indicates the maximum length of the data that can be stored. If the length is not specified, it means any length is available.

Example:

1. Create table **t2** and specify the character type of its columns.

```
CREATE TABLE t2 (a char(5),b varchar(5));
```

2. Insert data into table **t2** and query the byte length of column **a**. During table creation, the character type of column **a** is specified as **char(5)** and fixed-length. If the data length does not reach 5 bytes, spaces are added. Therefore, the queried byte length is **5**.

```
INSERT INTO t2 VALUES('abc','abc');
INSERT 0 1
```

```
SELECT a,lengthb(a),b FROM t2;
 a | lengthb | b
-----+-----
abc |      5 | abc
(1 row)
```

3. After the conversion by using the function, the actual queried length of the column **a** is **3** bytes.

```
SELECT a = b from t2;
?column?
```

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

```
SELECT cast(a as text) as val,lengthb(val) FROM t2;
 val | lengthb
-----+-----
abc |      3
(1 row)
```

Bytes and Characters

n means differently in **VARCHAR2(n)** and **NVARCHAR2(n)**.

- In **VARCHAR2(n)** **n** indicates the number of bytes.
- In **NVARCHAR2(n)**, **n** indicates the number of characters.

 NOTE

Take an UTF8-encoded database as an example. A letter occupies one byte, and a Chinese character occupies three bytes. **VARCHAR2(6)** allows for six letters or two Chinese characters, and **NVARCHAR2(6)** allows for six letters or six Chinese characters.

Empty Strings and NULL

In Oracle compatibility mode, empty strings and NULL are not distinguished. When a statement is executed to query or import data, empty strings are processed as NULL.

As such, `= ''` cannot be used as the query condition, and so does `is ''`. Otherwise, no result set is returned. The correct usage is **is null**, or **is not null**.

Example:

1. Create table **t4** and specify the character type of its columns.

```
CREATE TABLE t4 (a text);
```
2. Insert data into table **t4**. The inserted value contains an empty string and NULL.

```
INSERT INTO t4 VALUES('abc',''),(null);
INSERT 0 3
```
3. Check whether **t4** contains null values.

```
SELECT a,a isnull FROM t4;
a | ?column?
-----+-----
| t
| t
abc | f
(3 rows)

SELECT a,a isnull FROM t4 WHERE a is null;
a | ?column?
-----+-----
| t
| t
(2 rows)
```

In Teradata and MySQL compatibility mode, empty strings are distinguished from null.

- Example in Teradata compatibility mode:

```
SELECT '' is null , null is null;
isnull | isnull
-----+-----
f      | t
(1 rows)
```

- Example in MySQL compatibility mode:

```
SELECT '' is null , null is null;
isnull | isnull
-----+-----
f      | t
(1 rows)
```

21.5.5 Binary Data Types

Table 21-12 lists the binary data types that can be used in DWS.

Table 21-12 Binary Data Types

| Name | Description | Storage Space |
|-------|--|---|
| BLOB | <p>Binary large object.</p> <p>Currently, BLOB only supports the following external access interfaces:</p> <ul style="list-style-type: none">• DBMS_LOB.GETLENGTH• DBMS_LOB.READ• DBMS_LOB.WRITE• DBMS_LOB.WRITEAPPEND• DBMS_LOB.COPY• DBMS_LOB.ERASE <p>For details about these interfaces, see section "DBMS_LOB" in the <i>Data Warehouse Service (DWS) Developer Guide</i>.</p> <p>NOTE
Column storage cannot be used for the BLOB type.</p> | The maximum size is 10,7373,3621 bytes (1 GB - 8203 bytes). |
| RAW | <p>Variable-length hexadecimal string</p> <p>NOTE
Column storage cannot be used for the raw type.</p> | 4 bytes plus the actual hexadecimal string. The maximum size is 10,7373,3621 bytes (1 GB - 8203 bytes). |
| BYTEA | Variable-length binary string | 4 bytes plus the actual binary string. The maximum size is 10,7373,3621 bytes (1 GB - 8203 bytes). |

 NOTE

In addition to the size limitation on each column, the total size of each tuple is 8203 bytes less than 1 GB.

Examples

```
-- Create a table:
CREATE TABLE blob_type_t1
(
    BT_COL1 INTEGER,
    BT_COL2 BLOB,
    BT_COL3 RAW,
    BT_COL4 BYTEA
) DISTRIBUTE BY REPLICATION;

--Insert data:
INSERT INTO blob_type_t1 VALUES(10,empty_blob(),
HEXTORAW('DEADBEEF'),E'\xDEADBEEF');
```

```
-- Query data in the table:
SELECT * FROM blob_type_t1;
bt_col1 | bt_col2 | bt_col3 | bt_col4
-----+-----+-----+-----
      10 |      | DEADBEEF | \xdeadbeef
(1 row)

-- Delete the tables:
DROP TABLE blob_type_t1;
```

21.5.6 Date/Time Types

Table 21-13 lists date and time types supported by DWS. For the operators and built-in functions of the types, see [Date and Time Processing Functions and Operators](#).

NOTE

If the time format of another database is different from that described in DWS, modify the value of the **DateStyle** parameter to keep them consistent.

Table 21-13 Date/Time Types

| Name | Description | Storage Space |
|------------------------------------|--|--|
| DATE | In Oracle compatibility mode, it is equivalent to timestamp(0) and records the date and time.
In other modes, it records the date. | In Oracle compatibility mode, it occupies 8 bytes.
In Oracle compatibility mode, it occupies 4 bytes. |
| TIME [(p)] [WITHOUT TIME ZONE] | Specifies time within one day.
p indicates the precision after the decimal point. The value ranges from 0 to 6. | 8 bytes |
| TIME [(p)] [WITH TIME ZONE] | Specifies time within one day (with time zone).
p indicates the precision after the decimal point. The value ranges from 0 to 6. | 12 bytes |
| TIMESTAMP[(p)] [WITHOUT TIME ZONE] | Specifies the date and time.
p indicates the precision after the decimal point. The value ranges from 0 to 6. | 8 bytes |

| Name | Description | Storage Space |
|------------------------------------|---|---------------|
| TIMESTAMP[(p)]
[WITH TIME ZONE] | Specifies the date and time (with time zone). TIMESTAMP is also called TIMESTAMPTZ.

p indicates the precision after the decimal point. The value ranges from 0 to 6. | 8 bytes |
| SMALLDATETIME | Specifies the date and time (without time zone).

The precision level is minute. 31s to 59s are rounded into 1 minute. | 8 bytes |
| INTERVAL DAY (l)
TO SECOND (p) | Specifies the time interval (X days X hours X minutes X seconds). <ul style="list-style-type: none">• l: indicates the precision of days. The value ranges from 0 to 6. To adapt to Oracle syntax, the precision functions are not supported.• p: indicates the precision of seconds. The value ranges from 0 to 6. The digit 0 at the end of a decimal number is not displayed. | 16 bytes |

| Name | Description | Storage Space |
|------------------------------|--|---------------|
| INTERVAL [FIELDS]
[(p)] | <p>Specifies the time interval.</p> <ul style="list-style-type: none">• fields: YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, and MINUTE TO SECOND.• p: indicates the precision of seconds. The value ranges from 0 to 6. p takes effect only when fields are SECOND, DAY TO SECOND, HOUR TO SECOND, or MINUTE TO SECOND. The digit 0 at the end of a decimal number is not displayed. <p>NOTE
In MySQL compatibility mode of cluster version 9.1.0.200 and later, when the fields are YEAR, MONTH, DAY, HOUR, MINUTE, or SECOND, you can use an integer or floating point number as the input parameter (the number should not include positive or negative signs). For example, you can use INTERVAL 1 YEAR or INTERVAL 2.5 MONTH.</p> | 12 bytes |
| reltime | <p>Relative time interval. The format is:</p> <p><i>X</i> years <i>X</i> months <i>X</i> days <i>XX:XX:XX</i></p> <ul style="list-style-type: none">• The Julian calendar is used. It specifies that a year has 365.25 days and a month has 30 days. The relative time interval needs to be calculated based on the input value. The output format is POSTGRES. | 4 bytes |

Example:

```
--Create a table:
CREATE TABLE date_type_tab(coll date);

--Insert data:
INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data:
SELECT * FROM date_type_tab;
      coll
-----
2010-12-10 00:00:00
(1 row)
```



```
-- Delete the tables:
DROP TABLE date_type_tab;

--Create a table:
CREATE TABLE time_type_tab (da time without time zone ,dai time with time zone,dfgh timestamp without
time zone,dfga timestamp with time zone, vbg smalldatetime);

--Insert data:
INSERT INTO time_type_tab VALUES ('21:21:21','21:21:21 pst','2010-12-12','2013-12-11 pst','2003-04-12
04:05:06');

-- View data:
SELECT * FROM time_type_tab;
   da   |   dai   |   dfgh   |   dfga   |   vbg
-----+-----+-----+-----+-----
21:21:21 | 21:21:21-08 | 2010-12-12 00:00:00 | 2013-12-11 16:00:00+08 | 2003-04-12 04:05:00
(1 row)

-- Delete the tables:
DROP TABLE time_type_tab;

--Create a table:
CREATE TABLE day_type_tab (a int,b INTERVAL DAY(3) TO SECOND (4));

--Insert data:
INSERT INTO day_type_tab VALUES (1, INTERVAL '3' DAY);

-- View data:
SELECT * FROM day_type_tab;
 a | b
---+-----
 1 | 3 days
(1 row)

-- Delete the tables:
DROP TABLE day_type_tab;

--Create a table:
CREATE TABLE year_type_tab(a int, b interval year (6));

--Insert data:
INSERT INTO year_type_tab VALUES(1,interval '2' year);

-- View data:
SELECT * FROM year_type_tab;
 a | b
---+-----
 1 | 2 years
(1 row)

-- Delete the tables:
DROP TABLE year_type_tab;
```

Date Input

Date and time input is accepted in almost any reasonable formats, including ISO 8601, SQL-compatible, and traditional POSTGRES. The system allows you to customize the sequence of day, month, and year in the date input. Set the **DateStyle** parameter to **MDY** to select month-day-year interpretation, **DMY** to select day-month-year interpretation, or **YMD** to select year-month-day interpretation.

Remember that any date or time literal input needs to be enclosed with single quotes, and the syntax is as follows:

type [(p)] 'value'

The **p** that can be selected in the precision statement is an integer, indicating the number of fractional digits in the **seconds** column. [Table 21-14](#) shows some possible inputs for the **date** type.

Table 21-14 Date input

| Example | Description |
|------------------|--|
| 1999-01-08 | ISO 8601 (recommended format). January 8, 1999 in any mode |
| January 8, 1999 | Unambiguous in any date input mode |
| 1/8/1999 | January 8 in MDY mode. August 1 in DMY mode |
| 1/18/1999 | January 18 in MDY mode, rejected in other modes |
| 01/02/03 | <ul style="list-style-type: none">January 2, 2003 in MDY modeFebruary 1, 2003 in DMY modeFebruary 3, 2001 in YMD mode |
| 1999-Jan-08 | January 8 in any mode |
| Jan-08-1999 | January 8 in any mode |
| 08-Jan-1999 | January 8 in any mode |
| 99-Jan-08 | January 8 in YMD mode, else error |
| 08-Jan-99 | January 8, except error in YMD mode |
| Jan-08-99 | January 8, except error in YMD mode |
| 19990108 | ISO 8601. January 8, 1999 in any mode |
| 990108 | ISO 8601. January 8, 1999 in any mode |
| 1999.008 | Year and day of year |
| J2451187 | Julian date |
| January 8, 99 BC | Year 99 BC |

Example:

```
--Create a table:
CREATE TABLE date_type_tab(coll date);

--Insert data:
INSERT INTO date_type_tab VALUES (date '12-10-2010');

-- View data:
SELECT * FROM date_type_tab;
      coll
-----
2010-12-10 00:00:00
(1 row)

-- View the date format:
```

```
SHOW datestyle;
DateStyle
-----
ISO, MDY
(1 row)

-- Configure the date format:
SET datestyle='YMD';
SET

-- Insert data:
INSERT INTO date_type_tab VALUES(date '2010-12-11');

-- View data:
SELECT * FROM date_type_tab;
      coll
-----
2010-12-10 00:00:00
2010-12-11 00:00:00
(2 rows)

-- Delete the tables:
DROP TABLE date_type_tab;
```

Times

The time-of-day types are **TIME [(p)] [WITHOUT TIME ZONE]** and **TIME [(p)] [WITH TIME ZONE]**. **TIME** alone is equivalent to **TIME WITHOUT TIME ZONE**.

If a time zone is specified in the input for **TIME WITHOUT TIME ZONE**, it is silently ignored.

For details about the time input types, see [Table 21-15](#). For details about time zone input types, see [Table 21-16](#).

Table 21-15 Time input

| Example | Description |
|----------------|---|
| 05:06.8 | ISO 8601 |
| 4:05:06 | ISO 8601 |
| 4:05 | ISO 8601 |
| 40506 | ISO 8601 |
| 4:05 AM | Same as 04:05. AM does not affect value |
| 4:05 PM | Same as 16:05. Input hour must be <= 12 |
| 04:05:06.789-8 | ISO 8601 |
| 04:05:06-08:00 | ISO 8601 |
| 04:05-08:00 | ISO 8601 |
| 040506-08 | ISO 8601 |
| 04:05:06 PST | Time zone specified by abbreviation |

Table 21-16 Time zone input

| Example | Description |
|---------|--|
| PST | Abbreviation (for Pacific Standard Time) |
| -8:00 | ISO-8601 offset for PST |
| -800 | ISO-8601 offset for PST |
| -8 | ISO-8601 offset for PST |

Example:

```
SELECT time '04:05:06';
time
-----
04:05:06
(1 row)

SELECT time '04:05:06 PST';
time
-----
04:05:06
(1 row)

SELECT time with time zone '04:05:06 PST';
timetz
-----
04:05:06-08
(1 row)
```

Special Values

The special values supported by DWS are converted to common date/time values when being read. For details, see [Table 21-17](#).

Table 21-17 Special Values

| Input String | Applicable Type | Description |
|--------------|-----------------------|--|
| epoch | date, timestamp | 1970-01-01 00:00:00+00 (Unix system time zero) |
| infinity | timestamp | Later than any other timestamps |
| -infinity | timestamp | Earlier than any other timestamps |
| now | date, time, timestamp | Start time of the current transaction |
| today | date, timestamp | Today midnight |
| tomorrow | date, timestamp | Tomorrow midnight |
| yesterday | date, timestamp | Yesterday midnight |

| Input String | Applicable Type | Description |
|--------------|-----------------|-----------------|
| allballs | time | 00:00:00.00 UTC |

Interval Input

The input of **reltime** can be any valid interval in TEXT format. It can be a number (negative numbers and decimals are also allowed) or a specific time, which must be in SQL standard format, ISO-8601 format, or POSTGRES format. In addition, the text input needs to be enclosed with single quotation marks (').

For details, see [Table 21-18](#).

Table 21-18 Interval input

| Input | Output | Description |
|--------------------------------|-------------------------------------|---|
| 60 | 2 mons | Numbers are used to indicate intervals. The default unit is day. Decimals and negative numbers are also allowed. Particularly, a negative interval syntactically means how long before. |
| 31.25 | 1 mons 1 days 06:00:00 | |
| -365 | -12 mons -5 days | |
| 1 years 1 mons 8 days 12:00:00 | 1 years 1 mons 8 days 12:00:00 | Intervals are in POSTGRES format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input. |
| -13 months -10 hours | -1 years -25 days -04:00:00 | |
| -2 YEARS +5 MONTHS 10 DAYS | -1 years -6 mons -25 days -06:00:00 | |
| P-1.1Y10M | -3 mons -5 days -06:00:00 | Intervals are in ISO-8601 format. They can contain both positive and negative numbers and are case-insensitive. Output is a simplified POSTGRES interval converted from the input. |
| -12H | -12:00:00 | |

Example:

```
-- Create a table.
CREATE TABLE reltime_type_tab(col1 character(30), col2 reltime);

-- Insert data.
INSERT INTO reltime_type_tab VALUES ('90', '90');
INSERT INTO reltime_type_tab VALUES ('-366', '-366');
INSERT INTO reltime_type_tab VALUES ('1975.25', '1975.25');
INSERT INTO reltime_type_tab VALUES ('-2 YEARS +5 MONTHS 10 DAYS', '-2 YEARS +5 MONTHS 10 DAYS');
```

```
INSERT INTO reltime_type_tab VALUES ('30 DAYS 12:00:00', '30 DAYS 12:00:00');
INSERT INTO reltime_type_tab VALUES ('P-1.1Y10M', 'P-1.1Y10M');

-- View data.
SELECT * FROM reltime_type_tab;
      col1          |          col2
-----+-----
1975.25             | 5 years 4 mons 29 days
-2 YEARS +5 MONTHS 10 DAYS | -1 years -6 mons -25 days -06:00:00
P-1.1Y10M           | -3 mons -5 days -06:00:00
-366                | -1 years -18:00:00
90                  | 3 mons
30 DAYS 12:00:00    | 1 mon 12:00:00
(6 rows)

-- Delete tables.
DROP TABLE reltime_type_tab;
```

21.5.7 Geometric Types

Table 21-19 lists the geometric types that can be used in DWS. The most fundamental type, the point, forms the basis for all of the other types.

Table 21-19 Geometric Type

| Name | Storage Space | Description | Representation |
|---------|---------------|----------------------------------|-------------------------------------|
| point | 16 bytes | Point on a plane | (x,y) |
| lseg | 32 bytes | Finite line segment | ((x1,y1),(x2,y2)) |
| box | 32 bytes | Rectangular Box | ((x1,y1),(x2,y2)) |
| path | 16+16n bytes | Closed path (similar to polygon) | ((x1,y1),...) |
| path | 16+16n bytes | Open path | [(x1,y1),...] |
| polygon | 40+16n bytes | Polygon (similar to closed path) | ((x1,y1),...) |
| circle | 24 bytes | Circle | <(x,y),r> (center point and radius) |

A rich set of functions and operators is available in DWS to perform various geometric operations, such as scaling, translation, rotation, and determining intersections. For details, see [Geometric Functions and Operators](#).

Points

Points are the fundamental two-dimensional building block for geometric types. Values of the **point** type are specified using either of the following syntaxes:

```
( x , y )
x , y
```

where x and y are the respective coordinates, as floating-point numbers.

Points are output using the first syntax.

Line Segments

Line segments (**lseg**) are represented by pairs of points. Values of the **lseg** type are specified using any of the following syntaxes:

```
[ ( x1 , y1 ) , ( x2 , y2 ) ]  
( ( x1 , y1 ) , ( x2 , y2 ) )  
( x1 , y1 ) , ( x2 , y2 )  
x1 , y1 , x2 , y2
```

where (x1,y1) and (x2,y2) are the end points of the line segment.

Line segments are output using the first syntax.

Rectangular Box

Boxes are represented by pairs of points that are opposite corners of the box. Values of the **box** type are specified using any of the following syntaxes:

```
(( x1 , y1 ) , ( x2 , y2 ) )  
( x1 , y1 ) , ( x2 , y2 )  
x1 , y1 , x2 , y2
```

where (x1,y1) and (x2,y2) are any two opposite corners of the box.

Boxes are output using the second syntax.

Any two opposite corners can be supplied on input, but in this order, the values will be reordered as needed to store the upper right and lower left corners.

Path

Paths are represented by lists of connected points. Paths can be open, where the first and last points in the list are considered not connected, or closed, where the first and last points are considered connected.

Values of the **path** type are specified using any of the following syntaxes:

```
[ ( x1 , y1 ) , ... , ( xn , yn ) ]  
( ( x1 , y1 ) , ... , ( xn , yn ) )  
( x1 , y1 ) , ... , ( xn , yn )  
( x1 , y1 , ... , xn , yn )  
x1 , y1 , ... , xn , yn
```

where the points are the end points of the line segments comprising the path. Square brackets ([]) indicate an open path, while parentheses (()) indicate a closed path. When the outermost parentheses are omitted, as in the third through fifth syntaxes, a closed path is assumed.

Paths are output using the first or second syntax.

Polygons

Polygons are represented by lists of points (the vertexes of the polygon). Polygons are very similar to closed paths, but are stored differently and have their own set of support functions.

Values of the **polygon** type are specified using any of the following syntaxes:

```
(( x1 , y1 ) , ... , ( xn , yn ) )  
( x1 , y1 ) , ... , ( xn , yn )  
( x1 , y1 , ... , xn , yn )  
x1 , y1 , ... , xn , yn
```

where the points are the end points of the line segments comprising the boundary of the polygon.

Polygons are output using the first syntax.

Circle

Circles are represented by a center point and radius. Values of the **circle** type are specified using any of the following syntaxes:

```
< ( x , y ) , r >  
(( x , y ) , r )  
( x , y ) , r  
x , y , r
```

where **(x,y)** is the center point and **r** is the radius of the circle.

Circles are output using the first syntax.

21.5.8 Array

An array is a set of data. The array type allows a single database field to have multiple values. It is usually used to store and process data with similar attributes.

Syntax

```
ARRAY [ param ]
```

or

```
'{ param }'
```

The **param** parameter is described as follows:

- **param**: value contained in the array. It can be zero or many. Multiple values are separated by commas (.). If there is no value, set this parameter to NULL.
- When you specify an array constant in '{ *param* }' format, the elements of the string type cannot start or end with a single quotation mark. Instead, a double quotation mark is needed. Two consecutive single quotation marks will be converted to one single quotation mark.
- The data type of the first element is used as the data type of the array. All elements in an array must be of the same type, or their types can be converted to each other.

Definition of the Array Type

An array type is defined by adding values in the square brackets ([]) at the end of a data type.

Create the **books** table. The type of the **price** column (book prices) is a one-dimensional integer array, and the type of the **tags** column (book tags) is a two-dimensional text array.

```
CREATE TABLE books (id SERIAL PRIMARY KEY, title VARCHAR(100), price_by_quarter int[], tags TEXT[][]);
```


The **CREATE TABLE** syntax can specify the array size. For example:

```
CREATE TABLE test ( a int[3]);
```

In current database implementation, the array size limits of statements are ignored. Statements with array size limits behave the same as those without. At the same time, the number of declared dimensions is not required. Arrays of a particular element type are all treated as the same type. Their size or number of dimensions are ignored.

You can also use the keyword **ARRAY** to define a one-dimensional array. The **price** column in the **books** table is defined using **ARRAY** and the array size is specified as follows:

```
price_by_quarter int ARRAY[4]
```

Use **ARRAY** without specifying the array size:

```
price_by_quarter int ARRAY
```

Array Value Input

When entering an array value, write each array value as a literal constant, surround element values with braces, and separate them with commas. The general format of an array constant is as follows:

```
'{ val1 delim val2 delim ... }'
```

delim indicates a delimiter, and each **val** may be a constant or a subarray of an array element type.

An example of an array constant is as follows:

```
'{{1,2,3},{4,5,6},{7,8,9}}'
```

The constant is a two-dimensional, 3-by-3 array consisting of three integer subarrays.

Insert data into the **books** table and query the **books** table.

```
INSERT INTO books
VALUES (1, 'One Hundred years of Solitude', {25,25,25,25}, {'fiction'}, {adventure}),
      (2, 'Robinson Crusoe', {30,32,32,32}, {'adventure'}, {fiction}),
      (3, 'Gone with the Wind', {27,27,29,28}, {'romance'}, {fantasy});
```

```
SELECT * FROM books;
id | title | price_by_quarter | tags
---+-----+-----+-----
1 | One Hundred years of Solitude | {25,25,25,25} | {{fiction},{adventure}}
2 | Robinson Crusoe | {30,32,32,32} | {{adventure},{fiction}}
3 | Gone with the Wind | {27,27,29,28} | {{romance},{fantasy}}
(3 rows)
```

CAUTION

When multi-dimensional array data is inserted, each dimension of the array must have a matching length.

Use the **ARRAY** keyword to insert data.

```
INSERT INTO books
VALUES (1, 'One Hundred years of Solitude', ARRAY[25,25,25,25], ARRAY['fiction', 'adventure']),
```

```
(2, 'Robinson Crusoe', ARRAY[30,32,32,32], ARRAY['adventure', 'fiction']),  
(3, 'Gone with the Wind', ARRAY[27,27,29,28], ARRAY['romance', 'fantasy']);
```

Accessing Arrays

Accessing Array Elements

Query the names of the books whose prices change in the second quarter.

```
SELECT title FROM books WHERE price_by_quarter[1] <> price_by_quarter[2];  
title  
-----  
Robinson Crusoe  
(1 row)
```

Search for the prices of all books in the third quarter:

```
SELECT price_by_quarter[3] FROM books;  
price_by_quarter  
-----  
29  
25  
32  
(3 rows)
```

Accessing Array Slices

Any rectangular slice or subarray of an array can be accessed. An array slice can be defined by specifying [lower bound: upper bound] on one or more array dimensions.

Query the second label of **Gone with the Wind**.

```
SELECT tags[2:2] FROM books WHERE title = 'Gone with the Wind';  
tags  
-----  
{fantasy}  
(1 row)
```

Using Functions to Access Arrays

Use the **array_dims** function to obtain the dimension of an array value.

```
SELECT array_dims(tags) FROM books WHERE title = 'Robinson Crusoe';  
array_dims  
-----  
[1:2]  
(1 row)
```

You can also use **array_upper** and **array_lower** to obtain the array dimension. They return the upper and lower bounds of a specified array, respectively.

```
SELECT array_upper(tags, 1) FROM books WHERE title = 'Robinson Crusoe';  
array_upper  
-----  
2  
(1 row)
```

The **array_length** function returns the length of a specified array dimension.

```
SELECT array_length(tags, 1) FROM books WHERE title = 'Robinson Crusoe';  
array_length  
-----  
2  
(1 row)
```

Modifying an Array

Updating an Array

Update the entire array data:

```
UPDATE books SET price_by_quarter = '{30,30,30,30}'  
WHERE title = 'Robinson Crusoe';
```

Use the ARRAY expression to update the entire array data:

```
UPDATE books SET price_by_quarter = ARRAY[30,30,30,30]  
WHERE title = 'Robinson Crusoe';
```

Update an element in the array:

```
UPDATE books SET price_by_quarter[4] = 35  
WHERE title = 'Robinson Crusoe';
```

Update a slice element in the array:

```
UPDATE books SET price_by_quarter[1:2] = '{27,27}'  
WHERE title = 'Robinson Crusoe';
```

A stored array value can be expanded by assigning a value to a new element. The position between an existing element and the new element is filled with null values. For example, if the array **myarray** currently has four elements, it will have six elements after a value is assigned to **myarray[6]** using **UPDATE**. **myarray[5]** is filled with null. Currently, this method can be used only on one-dimensional arrays.

Building a New Array

New arrays can also be constructed using the concatenation operator ||. The concatenation operator allows a single element to be added to the beginning or end of a one-dimensional array. It can also accept two *N* dimensional arrays, or an *N* dimensional array and an *N*+1 dimensional array.

```
SELECT ARRAY[1,2] || ARRAY[3,4];  
?column?  
-----  
{1,2,3,4}  
(1 row)  
  
SELECT ARRAY[5,6] || ARRAY[[1,2],[3,4]];  
?column?  
-----  
{{5,6},{1,2},{3,4}}  
(1 row)
```

Use the **array_prepend**, **array_append**, or **array_cat** function to build an array.

```
SELECT array_prepend(1, ARRAY[2,3]);  
array_prepend  
-----  
{1,2,3}  
(1 row)  
  
SELECT array_append(ARRAY[1,2], 3);  
array_append  
-----  
{1,2,3}  
(1 row)  
  
SELECT array_cat(ARRAY[1,2], ARRAY[3,4]);  
array_cat  
-----
```

```
{1,2,3,4}
(1 row)

SELECT array_cat(ARRAY[[1,2],[3,4]], ARRAY[5,6]);

   array_cat
-----
{{1,2},{3,4},{5,6}}
(1 row)

SELECT array_cat(ARRAY[5,6], ARRAY[[1,2],[3,4]]);

   array_cat
-----
{{5,6},{1,2},{3,4}}
(1 row)
```

21.5.9 Enumeration Type

An enumeration (enum) type is a data type consisting of a static, ordered set of values. They are equivalent to the enum types used in many programming languages. The enumeration type can be a date in a week or a set of status values.

Declaration of Enumeration Types

Enumeration types can be created using the **CREATE TYPE** command. For example:

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
```

After an enumeration type is created, it can be used in table and function definitions.

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (name text, current_mood mood);
INSERT INTO person VALUES ('Moe', 'happy');
SELECT * FROM person WHERE current_mood = 'happy';
 name | current_mood
-----+-----
 Moe  | happy
(1 row)
```

Sorting

The order of enumeration values is the order of the values listed when the type is created.

```
INSERT INTO person VALUES ('Larry', 'sad');
INSERT INTO person VALUES ('Curly', 'ok');

SELECT * FROM person WHERE current_mood > 'sad';
 name | current_mood
-----+-----
 Moe  | happy
Curly | ok
(2 rows)

SELECT * FROM person WHERE current_mood > 'sad' ORDER BY current_mood;
 name | current_mood
-----+-----
Curly | ok
 Moe  | happy
(2 rows)

SELECT name FROM person WHERE current_mood = (SELECT MIN(current_mood) FROM person);
```

```
name
-----
Larry
(1 row)
```

Security of Enumeration Types

Each enumeration data type is independent and cannot be compared with other enumeration types.

```
CREATE TYPE happiness AS ENUM ('happy', 'very happy', 'ecstatic');
CREATE TABLE holidays (num_weeks integer, happiness happiness);
INSERT INTO holidays(num_weeks,happiness) VALUES (4, 'happy');
INSERT INTO holidays(num_weeks,happiness) VALUES (6, 'very happy');
INSERT INTO holidays(num_weeks,happiness) VALUES (8, 'ecstatic');

INSERT INTO holidays(num_weeks,happiness) VALUES (2, 'sad');
ERROR: invalid input value for enum happiness: "sad"

SELECT person.name, holidays.num_weeks FROM person, holidays
       WHERE person.current_mood = holidays.happiness;
ERROR: operator does not exist: mood = happiness
```

If comparison is required, you can use a customized operator or add an explicit type to the query.

```
SELECT person.name, holidays.num_weeks FROM person, holidays
       WHERE person.current_mood::text = holidays.happiness::text;
name | num_weeks
-----+-----
Moe  |         4
(1 row)
```

Precautions

- Enumeration tags are case sensitive. 'happy' is different from 'HAPPY'. Spaces in tags also have meanings.
- Although enumeration types are primarily intended for static collections of values, there are ways to add new and renamed values to existing enumeration types (see [ALTER TYPE](#)). Existing values cannot be removed from an enumeration type, and the order of these values cannot be changed unless the enumeration type is deleted and recreated.
- The mapping between enumerated values and text tags is stored in the **PG_ENUM** system catalog.

21.5.10 Network Address Types

DWS offers data types to store IPv4, IPv6, and MAC addresses.

It is better to use network address types instead of plaintext types to store IPv4, IPv6, and MAC addresses, because these types offer input error checking and specialized operators and functions. For details, see [Network Address Functions and Operators](#).

Table 21-20 Network Address Types

| Name | Storage Space | Description |
|------|---------------|-----------------------|
| cidr | 7 or 19 bytes | IPv4 or IPv6 networks |

| Name | Storage Space | Description |
|---------|---------------|---------------------------------|
| inet | 7 or 19 bytes | IPv4 or IPv6 hosts and networks |
| macaddr | 6 bytes | MAC addresses |

When sorting **inet** or **cidr** data types, IPv4 addresses will always sort before IPv6 addresses, including IPv4 addresses encapsulated or mapped to IPv6 addresses, such as ::10.2.3.4 or ::ffff:10.4.3.2.

cidr

The **cidr** type (Classless Inter-Domain Routing) holds an IPv4 or IPv6 network specification. The format for specifying networks is **address/y** where **address** is the network represented as an IPv4 or IPv6 address, and **y** is the number of bits in the netmask. If **y** is omitted, it is calculated using assumptions from the older classful network numbering system, except it will be at least large enough to include all of the octets written in the input.

- Example 1: Convert a value in the CIDR format to an IP address segment.

For example, **10.0.0.0/8** is converted into a 32-bit binary address **00001010.00000000.00000000.00000000**. **/8** indicates an 8-bit network ID. The first eight bits of the 32-bit binary address are fixed. The corresponding network segment is 00001010.00000000.00000000-00001010.11111111.11111111.11111111. **10.0.0.0/8** indicates that the subnet mask is 255.0.0.0 and the corresponding network segment is 10.0.0.0-10.255.255.255.

- Example 2: Convert an IP address segment to the CIDR format.

For the IP address segment **192.168.0.0–192.168.31.255**, the last two segments can be converted into a binary address **00000000.00000000-00011111.11111111**. The first 19 bits ($8 \times 2 + 3$) are fixed. Therefore, the binary address is 192.168.0.0/19.

Table 21-21 cidr type input examples

| cidr Input | cidr Output | abbrev (cidr) |
|--------------------|--------------------|--------------------|
| 192.168.100.128/25 | 192.168.100.128/25 | 192.168.100.128/25 |
| 192.168/24 | 192.168.0.0/24 | 192.168.0/24 |
| 192.168/25 | 192.168.0.0/25 | 192.168.0.0/25 |
| 192.168.1 | 192.168.1.0/24 | 192.168.1/24 |
| 192.168 | 192.168.0.0/24 | 192.168.0/24 |
| 10.1.2 | 10.1.2.0/24 | 10.1.2/24 |
| 10.1 | 10.1.0.0/16 | 10.1/16 |
| 10 | 10.0.0.0/8 | 10/8 |

| cidr Input | cidr Output | abbrev (cidr) |
|--|--|--------------------------------------|
| 10.1.2.3/32 | 10.1.2.3/32 | 10.1.2.3/32 |
| 2001:4f8:3:ba::/64 | 2001:4f8:3:ba::/64 | 2001:4f8:3:ba::/64 |
| 2001:4f8:3:ba:2e0:81ff:fe
22:d1f1/128 | 2001:4f8:3:ba:2e0:81ff:fe
22:d1f1/128 | 2001:4f8:3:ba:2e0:81ff:fe
22:d1f1 |

inet

The **inet** type holds an IPv4 or IPv6 host address, and optionally its subnet, all in one field. The subnet is represented by the number of network address bits present in the host address (the "netmask"). If the netmask is 32 and the address is IPv4, then the value does not indicate a subnet, only a single host. In IPv6, the address length is 128 bits, so 128 bits specify a unique host address.

The input format for this type is **address/y** where **address** is an IPv4 or IPv6 address and **y** is the number of bits in the netmask. If the **/y** portion is missing, the netmask is 32 for IPv4 and 128 for IPv6, so the value represents just a single host. On display, the **/y** portion is suppressed if the netmask specifies a single host.

The essential difference between the **inet** and **cidr** data types is that **inet** accepts values with nonzero bits to the right of the netmask, whereas **cidr** does not.

macaddr

The **macaddr** type stores MAC addresses, known for example from Ethernet card hardware addresses (although MAC addresses are used for other purposes as well). Input is accepted in the following formats:

```
'08:00:2b:01:02:03'  
'08-00-2b-01-02-03'  
'08002b:010203'  
'08002b-010203'  
'0800.2b01.0203'  
'08002b010203'
```

These examples would all specify the same address. Upper and lower cases are accepted for the digits a through f. Output is always in the first of the forms shown.

21.5.11 Bit String Types

Bit strings are strings of 1's and 0's. They can be used to store bit masks.

DWS supports two SQL bit types: **bit(n)** and **bit varying(n)**, where **n** is a positive integer.

The **bit** type data must match the length **n** exactly. It is an error to attempt to store shorter or longer bit strings. The **bit varying** data is of variable length up to the maximum length **n**; longer strings will be rejected. Writing **bit** without a length is equivalent to **bit(1)**, while **bit varying** without a length specification means unlimited length.

 NOTE

If one explicitly casts a bit-string value to **bit(n)**, it will be truncated or zero-padded on the right to be exactly **n** bits, without raising an error. Similarly, if one explicitly casts a bit-string value to **bit varying(n)**, the bits after **n** places will be truncated.

The following is an example of using the bit string type:

1. Create a sample table **bit_type_t1**.

```
CREATE TABLE bit_type_t1
(
  BT_COL1 INTEGER,
  BT_COL2 BIT(3),
  BT_COL3 BIT VARYING(5)
) DISTRIBUTE BY REPLICATION;
```

2. Insert data:

```
INSERT INTO bit_type_t1 VALUES(1, B'101', B'00');
```

An error is reported if an inserted string exceeds the length of this data type.

```
INSERT INTO bit_type_t1 VALUES(2, B'10', B'101');
ERROR: bit string length 2 does not match type bit(3)
CONTEXT: referenced column: bt_col2
```

3. Data is converted if it exceeds the length of this data type.

```
INSERT INTO bit_type_t1 VALUES(2, B'10'::bit(3), B'101');
```

4. View data:

```
SELECT * FROM bit_type_t1;
bt_col1 | bt_col2 | bt_col3
-----+-----
1 | 101 | 00
2 | 100 | 101
(2 rows)
```

21.5.12 Text Search Types

DWS offers **tsvector** and **tsquery** data types to support full text search. The **tsvector** type represents a document in a form optimized for text search. The **tsquery** type similarly represents a text query.

tsvector

The **tsvector** type represents a retrieval unit, usually a row of text fields in a database table or a combination of these fields.

A **tsvector** value is a sorted list of distinct lexemes, which are words that have been normalized to merge different variants of the same word. Sorting and duplicate-elimination are done automatically during input.

The **to_tsvector** function is used to parse and normalize a document string.

Use **tsvector** to segment a string into lexemes by space. The lexemes are sorted by letter and length. The following is an example:

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector;
tsvector
-----
'a' 'and' 'ate' 'cat' 'fat' 'mat' 'on' 'rat' 'sat'
(1 row)
```

To represent lexemes containing whitespace or punctuation, surround them with quotes:

```
SELECT '$the lexeme ' ' contains spaces$'::tsvector;
tsvector
```



```
-----  
' ' 'contains' 'lexeme' 'spaces' 'the'  
(1 row)
```

If a string is enclosed in common single quotation marks, the single quotation marks (') and backslashes (\) embedded in the string must be double-written for escape.

```
SELECT $$the lexeme 'Joe's' contains a quote$$::tsvector;  
tsvector  
-----  
'Joe's' 'a' 'contains' 'lexeme' 'quote' 'the'  
(1 row)
```

Optionally, integer positions can be attached to lexemes:

```
SELECT 'a:1 fat:2 cat:3 sat:4 on:5 a:6 mat:7 and:8 ate:9 a:10 fat:11 rat:12'::tsvector;  
tsvector  
-----  
'a':1,6,10 'and':8 'ate':9 'cat':3 'fat':2,11 'mat':7 'on':5 'rat':12 'sat':4  
(1 row)
```

A position normally indicates the source word's location in the document. Positional information can be used for proximity ranking. Position values range from 1 to 16383. The default maximum value is **16383**. Duplicate positions for the same lexeme are discarded.

Lexemes that have positions can further be labeled with a weight, which can be **A**, **B**, **C**, or **D**. **D** is the default weight. It is not displayed in the output:

```
SELECT 'a:1A fat:2B,4C cat:5D'::tsvector;  
tsvector  
-----  
'a':1A 'cat':5 'fat':2B,4C  
(1 row)
```

Weights usually are used to reflect document structure, for example, by marking title words differently from body words. Text search ranking functions can assign different priorities to the different weight markers.

The following is an example of the standard usage of the tsvector type:

```
SELECT 'The Fat Rats'::tsvector;  
tsvector  
-----  
'Fat' 'Rats' 'The'  
(1 row)
```

For most English-text-searching applications the above words would be considered non-normalized, which should usually be passed through **to_tsvector** to normalize the words appropriately for searching:

```
SELECT to_tsvector('english', 'The Fat Rats');  
to_tsvector  
-----  
'fat':2 'rat':3  
(1 row)
```

tsquery

The **tsquery** type represents a retrieval condition. A **tsquery** value stores lexemes that are to be searched for, and combines them honoring the **Boolean** operators **&** (**AND**), **|** (**OR**), and **!** (**NOT**). Parentheses can be used to enforce grouping of the operators. If there is no parenthesis, (**NOT**) has the highest priority, followed by

&(AND), and finally **| (OR)**. The **to_tsquery** and **plainto_tsquery** functions will normalize lexemes before the lexemes are converted to the **tsquery** type.

```
SELECT 'fat & rat'::tsquery;
      tsquery
-----
'fat' & 'rat'
(1 row)

SELECT 'fat & (rat | cat)'::tsquery;
      tsquery
-----
'fat' & ( 'rat' | 'cat' )
(1 row)

SELECT 'fat & rat & ! cat'::tsquery;
      tsquery
-----
'fat' & 'rat' & '!cat'
(1 row)
```

Lexemes in a **tsquery** can be labeled with one or more weight letters, which match only **tsvector** lexemes with matching weights.

```
SELECT 'fat:ab & cat'::tsquery;
      tsquery
-----
'fat':AB & 'cat'
(1 row)
```

Also, lexemes in a **tsquery** can be labeled with ***** to specify prefix matching. The following query will match any word in a **tsvector** that begins with "super".

```
SELECT 'super:*'::tsquery;
      tsquery
-----
'super':*
(1 row)
```

Note that prefix matches are first checked by the text search analyzer. For example, the stem extracted from **postgres** is **postgr**, which matches **postgraduate**. The following result is true:

```
SELECT to_tsvector( 'postgraduate' ) @@ to_tsquery( 'postgres:*' ) AS RESULT;
      result
-----
t
(1 row)

SELECT to_tsquery('postgres:*');
      to_tsquery
-----
'postgr':*
(1 row)
```

The **to_tsquery** function normalizes words before converting them to the **tsquery** type. **'Fat:ab & Cats'** is normalized to the **tsquery** type as follows:

```
SELECT to_tsquery('Fat:ab & Cats');
      to_tsquery
-----
'fat':AB & 'cat'
(1 row)
```

21.5.13 UUID Type

Universally Unique Identifier (UUID) is a 128-bit identifier used in the computer system to identify information.

All elements in a distributed system can be uniquely identified by an UUID, without having to be identified in the central control end. In many application scenarios, an ID is required to identify only one object. A common example is the ID column of a database table. Another example is the front end UI libraries, because they usually need to dynamically create various UI elements that require unique IDs.

For details about UUID functions and UUID application examples supported by DWS, see [UUID Functions](#).

UUID Formats

UUIDs are standardized as part of the Distributed Computing Environment (DCE) in RFC 4122 released by the Internet Engineering Task Force (IETF) of the Open Software Foundation (OSF). A standard UUID consists of 36 characters, including 32 hexadecimal digits and four hyphens (-). The format is 8-4-4-4-12. An example of a standard UUID is as follows:

```
a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11
```

DWS also accepts the following alternative forms for input: use of upper-case letters and digits, the standard format surrounded by braces, omitting some or all hyphens, and adding a hyphen after any group of four digits. Examples

```
A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11  
{a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11}  
a0eebc99c0b4ef8bb6d6bb9bd380a11  
a0ee-bc99-9c0b-4ef8-bb6d-6bb9-bd38-0a11
```

UUID Composition

A UUID uses time and space information to ensure global uniqueness. A UUID consists of a timestamp, a clock sequence, and a node ID (MAC address). However, multiple nodes in a distributed DWS cluster may be deployed on the same machine and have the same MAC address. As a result, UUID conflicts may occur. Therefore, DWS replaces the last 48 bits of the MAC address with the sequence number of the CN or DN that generates the UUID and the current thread ID to ensure that the UUID is globally unique in the distributed cluster.

21.5.14 JSON Types

JavaScript Object Notation (JSON) data types are used for storing JSON data.

It can be an independent scalar, an array, or a key-value object. An array and an object can be called a container.

1. Scalar: a number, Boolean, string, or null
2. Array: defined in a pair of square brackets ([]), in which elements can be of any JSON data type, and are not necessarily of the same type.
3. Object: defined in a pair of braces ({}), in which objects are stored in the format of **key:value**. Each key must be a string enclosed in double quotation marks (""), and its value can be of any JSON data type. In case of duplicate keys, the last key-value pair will be used.

DWS supports the JSON and JSONB data types to store JSON data. Where:

- json copies all entered strings and parses them when they are used. During this process, the entered spaces, duplicate keys, and sequence are retained.
- jsonb parses the binary data of the input. During parsing, jsonb deletes semantic-irrelevant details and duplicate keys, and sorts key values, so that the data does not to be parsed again during use.

Both JSON and JSONB are of JSON data type, and the same strings can be entered as input. The main difference between JSON and JSONB is the efficiency. Because json data is an exact copy of the input text, the data must be parsed on every execution. In contrast, jsonb data is stored in a decomposed binary form and can be processed faster, though this makes it slightly slower to input due to the conversion mechanism. In addition, because the JSONB data form is normalized, it supports more operations, for example, comparing sizes according to a specific rule. JSONB also supports indexing, which is a significant advantage.

Input Format

The input must be a JSON-compliant string, which is enclosed in single quotation marks (').

Null (null-json): Only null is supported, and all letters are in lowercase.

```
SELECT 'null':json; -- suc
SELECT 'NULL':jsonb; -- err
```

Number (num-json): The value can be a positive or negative integer, decimal fraction, or 0. The scientific notation is supported.

```
SELECT '1':json;
SELECT '-1.5':json;
SELECT '-1.5e-5':jsonb, '-1.5e+2':jsonb;
SELECT '001':json, '+15':json, 'NaN':json; -- Redundant leading zeros, plus signs (+), NaN, and infinity are not supported.
```

Boolean (bool-json): The value can only be **true** or **false** in lowercase.

```
SELECT 'true':json;
SELECT 'false':jsonb;
```

String (str-json): The value must be a string enclosed in double quotation marks (").

```
SELECT '"a"':json;
SELECT '"abc"':jsonb;
```

Array (array-json): Arrays are enclosed in square brackets ([]). Elements in the array can be any valid JSON data, and are unnecessarily of the same type.

```
SELECT '[1, 2, "foo", null]':json;
SELECT '[]':json;
SELECT '[1, 2, "foo", null, [], {}]':jsonb;
```

Object (object-json): The value is enclosed in braces ({}). The key must be a JSON-compliant string, and the value can be any valid JSON string.

```
SELECT '{}':json;
SELECT '{"a": 1, "b": {"a": 2, "b": null}}':json;
SELECT '{"foo": [true, "bar"], "tags": {"a": 1, "b": null}}':jsonb;
```

 CAUTION

- Note that **'null::jsonb** and **null::jsonb** are different. The difference is similar to that between the strings **str=""** and **str=null**.
- For numbers, when scientific notation is used, JSONB expands them, while JSON stores an exact copy of the input text.

JSONB Advanced Features

The main difference between JSON and JSONB is the storage mode. JSONB stores parsed binary data, which reflects the JSON hierarchy and facilitates direct access. Therefore, JSONB has more advanced features than JSON.

Normalizes formats

- After the input **object-json** string is parsed into JSONB binary, semantically irrelevant details are naturally discarded, for example, spaces:

```
SELECT ' [1, " a ", {"a" :1  }] '::jsonb;
jsonb
-----
[1, " a ", {"a": 1}]
(1 row)
```

- For **object-json**, duplicate key-values are deleted and only the last key-value is retained. An example is as follows:

```
SELECT '{"a" : 1, "a" : 2} '::jsonb;
jsonb
-----
{"a": 2}
(1 row)
```

- For **object-json**, key-values will be re-sorted. The sorting rule is as follows: 1. Longer key-values are sorted last. 2. If the key-values are of the same length, the key-values with a larger ASCII code are sorted last. An example is as follows:

```
SELECT '{"aa" : 1, "b" : 2, "a" : 3} '::jsonb;
jsonb
-----
{"a": 3, "b": 2, "aa": 1}
(1 row)
```

Compares sizes

Format normalization ensures that only one form of JSONB data exists in the same semantics. Therefore, sizes can be compared according to a specific rule.

1. First, type comparison: **object-jsonb** > **array-jsonb** > **bool-jsonb** > **num-jsonb** > **str-jsonb** > **null-jsonb**
2. Content is compared if the data type is the same:
 - **str-json**: The default text sorting rule of the database is used for comparison. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.
 - **num-json**: numeric comparison
 - **bool-json**: **true** > **false**
 - **array-jsonb**: long elements > short elements. If the lengths are the same, compare each element in sequence.

- **object-jsonb**: long key-value pairs > short key-value pairs. If the lengths are the same, compare each key-value pair in sequence, first the key and then the value.

CAUTION

For comparison within the **object-jsonb** type, the final result after format sorting is used for comparison. Therefore, the comparison result may not be so intuitive as direct input.

Creates an index

The JSONB type in row-store tables supports both B-tree and GIN indexes, while the JSONB type in column-store tables only supports B-tree indexes.

If the entire JSONB column uses a B-tree index, the following operators can be used: =, <, <=, >, and >=.

Example: Create the table **test** and insert data into it.

```
CREATE TABLE test(id bigserial, data JSONB, PRIMARY KEY (id));
INSERT INTO test(data) VALUES('{"name":"Jack", "age":10, "nick_name":["Jacky","baobao"], "phone_list":
["1111","2222"]}::jsonb);
```

- Create a B-tree index.

```
CREATE INDEX idx_test_data_age ON test USING btree(((data->>'age')::int));
```

Use the B-tree index to query data where "age">1".

```
SELECT * FROM test WHERE (data->>'age')::int>1;
```

- Create a GIN index.

```
CREATE INDEX idx_test_data ON test USING gin (data);
```

Use the GIN index to check whether there are top-level keywords.

```
SELECT * FROM test WHERE data ? 'id';
```

```
SELECT * FROM test WHERE data ?| array['id','name'];
```

- Use the GIN index to check whether there are non-top-level keywords.

```
CREATE INDEX idx_test_data_nick_name ON test USING gin((data->'nick_name'));
SELECT * FROM test WHERE data->'nick_name' ? 'Jacky';
```

- Use @> to check whether JSON contains nested JSON objects.

```
SELECT * FROM test WHERE data @> '{"age":10, "nick_name":["Jacky"]}';
```

Includes elements in a JSON

An important capability of JSONB is to query whether a JSON contains some elements or whether some elements exist in a JSON.

- Simple scalar/original values contain only the same value.

```
SELECT '"foo"'::jsonb @> '"foo"'::jsonb;
```

- The array on the left contains the string on the right.

```
SELECT '[1, "aa", 3]'::jsonb ? 'aa';
```

- The array on the left contains all elements of the array on the right. The sequence and repetition are not important.

```
SELECT '[1, 2, 3]'::jsonb @> '[1, 3, 1]'::jsonb;
```

- The **object-json** on the left contains all key-values of the **object-json** on the right.

```
SELECT '{"product": "PostgreSQL", "version": 9.4, "jsonb":true}'::jsonb @> '{"version":9.4}'::jsonb;
```

- The array on the left does not contain all elements in the array on the right, because the three elements in the array on the left are **1**, **2**, and **[1,3]**, and the elements on the right are **1** and **3**.

```
SELECT '[1, 2, [1, 3]]::jsonb @> '[1, 3]::jsonb; -- false
```
- The array on the right does not contain all elements in the array on the left in the following example:

```
SELECT '{"foo": {"bar": "baz"}}::jsonb @> '{"bar": "baz"}::jsonb; -- false
```

21.5.15 RoaringBitmap

In DWS 8.1.3 and later, you can use the RoaringBitmap data type to store bitmap datasets.

The RoaringBitmap data type supports row-store and column-store tables.

Table 21-22 RoaringBitmap

| Name | Storage Size | Description | Range |
|---------------|--------------|--------------------------|------------------------------|
| RoaringBitmap | 32 bytes | Stores bitmap data sets. | -2,147,483,648~2,147,483,647 |

Example: Create a table with the RoaringBitmap data type.

```
CREATE TABLE r_row (a int ,b text, c roaringbitmap);  
CREATE TABLE r_col (a int ,b text, c roaringbitmap) with (orientation=column);
```

21.5.16 HLL Data Types

HyperLoglog (HLL) is an approximation algorithm for efficiently counting the number of distinct values in a data set. It features faster computing and lower space usage. You only need to store HLL data structures, instead of data sets. When new data is added to a data set, make hash calculation on the data and insert the result to an HLL. Then, you can obtain the final result based on the HLL.

Table 21-23 compares HLL with other algorithms.

Table 21-23 Comparison between HLL and other algorithms

| Item | Sorting Algorithm | Hash Algorithm | HLL |
|---------------------------|-------------------|------------------|------------|
| Time complexity | O(nlogn) | O(n) | O(n) |
| Space complexity | O(n) | O(n) | 1280 bytes |
| Error rate | 0 | 0 | ≈2% |
| Storage space requirement | Size of raw data | Size of raw data | 1280 bytes |

HLL has advantages over others in the computing speed and storage space requirement. In terms of time complexity, the sorting algorithm needs $O(n \log n)$ time for sorting, and the hash algorithm and HLL need $O(n)$ time for full table scanning. In terms of storage space requirements, the sorting algorithm and hash algorithm need to store raw data before collecting statistics, whereas the HLL algorithm needs to store only the HLL data structures rather than the raw data, and thereby occupying a fixed space of only 1280 bytes.

NOTICE

- In default specifications, the maximum number of distinct values is 1.6e plus 12, and the maximum error rate is only 2.3%. If a calculation result exceeds the maximum number, the error rate of the calculation result will increase, or the calculation will fail and an error will be reported.
- When using this feature for the first time, you need to evaluate the distinct values of the service, properly select configuration parameters, and perform verification to ensure that the accuracy meets requirements.
 - When default parameter configuration is used, the calculated number of distinct values is 1.6e plus 12. If the calculated result is **NaN**, you need to adjust **log2m** and **regwidth** to accommodate more distinct values.
 - The hash algorithm has an extremely low probability of collision. However, you are still advised to select 2 or 3 hash seeds for verification when using the hash algorithm for the first time. If there is only a small difference between the distinct values, you can select any one of the seeds as the hash seed.

[Table 21-24](#) describes main HLL data structures.

Table 21-24 Main HLL data structures

| Data Type | Description |
|-----------|---|
| hll | Its size is always 1280 bytes, which can be directly used to calculate the number of distinct values. |

Application Scenarios of HLL

- Using the HLL Data Type
 - a. Create an HLL table and insert an empty HLL into the table.

```
CREATE TABLE helloworld (id integer, set hll);
INSERT INTO helloworld(id, set) VALUES (1, hll_empty());
```
 - b. Add an integer that has gone through hash calculation into to the HLL.

```
UPDATE helloworld SET set = hll_add(set, hll_hash_integer(12345)) WHERE id = 1;
```
 - c. Add a string that has gone through hash calculation into to the HLL.

```
UPDATE helloworld SET set = hll_add(set, hll_hash_text('hello world')) WHERE id = 1;
```
 - d. Obtain the number of distinct values of the HLL.

```
SELECT hll_cardinality(set) FROM helloworld WHERE id = 1;
hll_cardinality
-----
```



```

      2
(1 row)

```

- Using HLL to Collect Website Visitor Statistics

- Create the original data table **facts** to record the time when a user accesses a website.

```

CREATE TABLE facts (
  date      date,
  user_id   integer
);

```

- Insert user visits data.

```

INSERT INTO facts VALUES ('2019-02-20', generate_series(1,100));
INSERT INTO facts VALUES ('2019-02-21', generate_series(1,200));
INSERT INTO facts VALUES ('2019-02-22', generate_series(1,300));
INSERT INTO facts VALUES ('2019-02-23', generate_series(1,400));
INSERT INTO facts VALUES ('2019-02-24', generate_series(1,500));
INSERT INTO facts VALUES ('2019-02-25', generate_series(1,600));
INSERT INTO facts VALUES ('2019-02-26', generate_series(1,700));
INSERT INTO facts VALUES ('2019-02-27', generate_series(1,800));

```

- Create another table and specify an HLL column: Group data by date and insert the data into the HLL:

```

CREATE TABLE daily_uniques (
  date      date UNIQUE,
  users     hll
);

INSERT INTO daily_uniques(date, users)
SELECT date, hll_add_agg(hll_hash_integer(user_id))
FROM facts
GROUP BY 1;

```

- Calculate the numbers of users visiting the website every day:

```

SELECT date, hll_cardinality(users) FROM daily_uniques ORDER BY date;

```

| date | hll_cardinality |
|---------------------|------------------|
| 2019-02-20 00:00:00 | 100 |
| 2019-02-21 00:00:00 | 203.813355588808 |
| 2019-02-22 00:00:00 | 308.048239950384 |
| 2019-02-23 00:00:00 | 410.529188080374 |
| 2019-02-24 00:00:00 | 513.263875705319 |
| 2019-02-25 00:00:00 | 609.271181107416 |
| 2019-02-26 00:00:00 | 702.941844662509 |
| 2019-02-27 00:00:00 | 792.249946595237 |

(8 rows)

- Calculate the number of users who had visited the website in the week from February 20, 2019 to February 26, 2019:

```

SELECT hll_cardinality(hll_union_agg(users)) FROM daily_uniques WHERE date >=
'2019-02-20'::date AND date <= '2019-02-26'::date;

```

| hll_cardinality |
|------------------|
| 702.941844662509 |

(1 row)

- Calculate the number of users who had visited the website yesterday but have not visited the website today:

```

SELECT date, (#hll_union_agg(users) OVER two_days) - #users AS lost_uniques FROM
daily_uniques WINDOW two_days AS (ORDER BY date ASC ROWS 1
PRECEDING);

```

| date | lost_uniques |
|---------------------|--------------|
| 2019-02-20 00:00:00 | 0 |
| 2019-02-21 00:00:00 | 0 |
| 2019-02-22 00:00:00 | 0 |
| 2019-02-23 00:00:00 | 0 |
| 2019-02-24 00:00:00 | 0 |
| 2019-02-25 00:00:00 | 0 |

```
2019-02-26 00:00:00 | 0
2019-02-27 00:00:00 | 0
(8 rows)
```

- HLL Format Errors

When inserting data into a column of the HLL type, ensure that the data meets the requirements of the HLL data structure. If the data does not meet the requirements after being parsed, an error will be reported.

For example, when `E\1234` is inserted, the data does not comply with the HLL data structure and cannot be parsed. As a result, an error is reported.

```
CREATE TABLE test(id integer, set hll);
INSERT INTO test VALUES(1, 'E\1234');
ERROR: invalid input syntax for integer: "E\1234"
```

21.5.17 Object Identifier Types

Object identifiers (OIDs) are used internally by DWS as primary keys for various system catalogs. OIDs are not added to user-created tables by the system. The **OID** type represents an object identifier.

The **OID** type is currently implemented as an unsigned four-byte integer. So, using a user-created table's **OID** column as a primary key is discouraged.

Table 21-25 Object identifier types

| Name | Reference | Description | Examples |
|-----------|--------------|---|----------|
| OID | - | Numeric object identifier | 564182 |
| CID | - | A command identifier. This is the data type of the system columns cmin and cmax . Command identifiers are 32-bit quantities. | - |
| XID | - | A transaction identifier. This is the data type of the system columns xmin and xmax . Transaction identifiers are also 32-bit quantities. | - |
| TID | - | A row identifier. This is the data type of the system column ctid . A row ID is a pair (block number, tuple index within block) that identifies the physical location of the row within its table. | - |
| REGCONFIG | pg_ts_config | Text search configuration | english |

| Name | Reference | Description | Examples |
|---------------|-------------|-------------------------------------|---------------------------------------|
| REGDICTIONARY | pg_ts_dict | Text search dictionary | simple |
| REGOPER | pg_operator | Operator name | + |
| REGOPERATOR | pg_operator | Operator with argument types | *(integer,integer) or -(NONE,integer) |
| REGPROC | pg_proc | Indicates the name of the function. | sum |
| REGPROCEDURE | pg_proc | Function with argument types | sum(int4) |
| REGCLASS | pg_class | Relation name | pg_type |
| REGTYPE | pg_type | Data type name | integer |

The **OID** type is used for a column in the database system catalog.

Example:

```
SELECT oid FROM pg_class WHERE relname = 'pg_type';
oid
-----
1247
(1 row)
```

The alias type for **OID** is **REGCLASS** which allows simplified search for **OID** values.

Example:

```
SELECT attrelid,attname,atttypid,attstattarget FROM pg_attribute WHERE attrelid = 'pg_type'::REGCLASS;
attrelid | attname | atttypid | attstattarget
-----+-----+-----+-----
1247 | xc_node_id | 23 | 0
1247 | tableoid | 26 | 0
1247 | cmax | 29 | 0
1247 | xmax | 28 | 0
1247 | cmin | 29 | 0
1247 | xmin | 28 | 0
1247 | oid | 26 | 0
1247 | ctid | 27 | 0
1247 | typname | 19 | -1
1247 | typnamespace | 26 | -1
1247 | typowner | 26 | -1
1247 | typplen | 21 | -1
1247 | typbyval | 16 | -1
1247 | typtype | 18 | -1
1247 | typcategory | 18 | -1
1247 | typispreferred | 16 | -1
1247 | typisdefined | 16 | -1
1247 | typdelim | 18 | -1
1247 | typrelid | 26 | -1
1247 | typelem | 26 | -1
1247 | typarray | 26 | -1
1247 | typinput | 24 | -1
1247 | typoutput | 24 | -1
```

| | | | | | |
|-----------|---------------|--|------|--|----|
| 1247 | typreceive | | 24 | | -1 |
| 1247 | typsend | | 24 | | -1 |
| 1247 | typmodin | | 24 | | -1 |
| 1247 | typmodout | | 24 | | -1 |
| 1247 | typanalyze | | 24 | | -1 |
| 1247 | typalign | | 18 | | -1 |
| 1247 | typstorage | | 18 | | -1 |
| 1247 | typnotnull | | 16 | | -1 |
| 1247 | typbasetype | | 26 | | -1 |
| 1247 | typtypmod | | 23 | | -1 |
| 1247 | typndims | | 23 | | -1 |
| 1247 | typcollation | | 26 | | -1 |
| 1247 | typdefaultbin | | 194 | | -1 |
| 1247 | typdefault | | 25 | | -1 |
| 1247 | typacl | | 1034 | | -1 |
| (38 rows) | | | | | |

21.5.18 Pseudo-Types

DWS has a number of special-purpose entries that are collectively called pseudo-types. A pseudo-type cannot be used as a column data type, but it can be used to declare a function's argument or result type.

Pseudotypes are highly beneficial in functions that require more complex input and output than standard SQL data types. [Table 21-26](#) lists all pseudo-types.

Table 21-26 Pseudo-Types

| Name | Description |
|------------------|---|
| any | Indicates that a function accepts any input data type. |
| anyelement | Indicates that a function accepts any data type. |
| anyarray | Indicates that a function accepts any array data type. |
| anynonarray | Indicates that a function accepts any non-array data type. |
| anyenum | Indicates that a function accepts any enum data type. |
| anyrange | Indicates that a function accepts any range data type. |
| cstring | Indicates that a function accepts or returns a null-terminated C string. |
| internal | Indicates that a function accepts or returns a server-internal data type. |
| language_handler | Indicates that a procedural language call handler is declared to return language_handler . |
| fdw_handler | Indicates that a foreign-data wrapper handler is declared to return fdw_handler . |
| record | Identifies a function returning an unspecified row type. |
| trigger | Indicates that a trigger function is declared to return trigger . |
| void | Indicates that a function returns no value. |

| Name | Description |
|--------|--|
| opaque | Indicates an obsolete type name that formerly served all the above purposes. |

Functions coded in C (whether built in or dynamically loaded) can be declared to accept or return any of these pseudo data types. It is up to the function author to ensure that the function will behave safely when a pseudo-type is used as an argument type.

Functions coded in procedural languages can use pseudo-types only as allowed by their implementation languages. At present the procedural languages all forbid use of a pseudo-type as argument type, and allow only **void** and **record** as a result type. Some also support polymorphic functions using the **anyelement**, **anyarray**, **anynonarray**, **anyenum**, and **anyrange** types.

The **internal** pseudo-type is used to declare functions that are meant only to be called internally by the database system, and not by direct call in an SQL query. If a function has at least one **internal**-type argument, it cannot be called from SQL. You are not advised to create any function that is declared to return **internal** unless the function has at least one **internal** argument.

Example:

Create or replace the showall() function:

```
CREATE OR REPLACE FUNCTION showall() RETURNS SETOF record
AS $$ SELECT count(*) from tpeds.store_sales where ss_customer_sk = 9692; $$
LANGUAGE SQL;
```

Invoke the showall() function:

```
SELECT showall();
showall
-----
(35)
(1 row)
```

Delete a function.

```
DROP FUNCTION showall();
```

21.5.19 Range Types

A range type is a data type that represents a range of values for certain element types (subtypes of ranges). For example, the range of a timestamp may be used to express a time range in which a conference room is reserved. In this case, the data type is **tsrange** (short for timestamp range), and **timestamp** is the subtype. The subtype must have an overall order so that the element value can be clearly specified as within, before, or after a range.

Range types can express multiple element values in a single range value and can clearly express concepts such as range overlapping. It helps to schedule the timing and duration of an event, and can also be utilized to set price or measurement parameters for equipment.

Built-in Range Types

DWS has the following built-in range types:

- `int4range`: integer range.
- `int8range`: bigint range.
- `numrange`: numeric range.
- `tsrange`: range of timestamp without the time zone.
- `tstzrange`: range of timestamp with the time zone
- `daterange`: date range.

NOTE

DWS does not support user-defined range types.

Create the reservation table and insert data into the table. The **during** column is of the **tsrange** type.

```
CREATE TABLE reservation (room int, during tsrange);
INSERT INTO reservation VALUES (1108, '[2010-01-01 14:30, 2010-01-01 15:30)');
```

Query whether the range contains specific values:

```
SELECT int4range(10, 20) @> 3;
?column?
-----
f
(1 row)
```

Query whether the range overlaps with another value:

```
SELECT numrange(11.1, 22.2) && numrange(20.0, 30.0);
?column?
-----
t
(1 row)
```

Obtain the upper bound of the range:

```
SELECT upper(int8range(15, 25));
upper
-----
25
(1 row)
```

Calculate the intersection with another range:

```
SELECT int4range(10, 20) * int4range(15, 25);
?column?
-----
[15,20)
(1 row)
```

Query whether the range is empty:

```
SELECT isempty(numrange(1, 5));
isempty
-----
f
(1 row)
```

Inclusion and Exclusion of Bounds

Each non-empty range has two bounds: a lower bound and an upper bound. All values between the two bounds are included in the range. Inclusion of bounds

means that boundary values are also included in the range, while exclusion of bounds means that boundary values are not included in the range.

Inclusion of lower bound is represented by "[", and exclusion of lower bound is represented by "(". Similarly, inclusion of upper bound is represented by "]", and exclusion of upper bound is represented by ")".

The `lower_inc(anyrange)` and `lower_inc(anyrange)` functions are used to test the upper and lower bounds of a range, respectively.

Infinite (Unbounded) Range

When the lower bound of a range is unbounded, it means that all values less than the upper bound are included in the range, for example, `(,3]`. Similarly, when the upper bound of a range is unbounded, all values greater than the upper bound are included in the range. When both the upper and lower bounds are unbounded, all values of the element type are considered within the range. The missing bounds are automatically converted to exclusions, for example, `[,]` is converted to `(,)`. You can think of these missing values as positive infinity or negative infinity, but they are special range type values and are considered to be positive and negative infinity values that go beyond any range element type.

Element types with the infinity values can be used as explicit bound values. For example, in the timestamp range, `[today, infinity)` does not include a special timestamp value **infinity**.

The functions `lower_inf(anyrange)` and `upper_inf(anyrange)` are used to test whether a range has infinite upper bound or lower bound, respectively.

Input/Output Values of a Range

Range values must follow one of the following patterns:

```
(lower-bound,upper-bound)
(lower-bound,upper-bound]
[lower-bound,upper-bound)
[lower-bound,upper-bound]
empty
```

Parentheses `()` or square brackets `[]` indicate whether the upper and lower bounds are excluded or included. The last format is **empty**, which represents an empty range (a range that does not contain values).

The value of *lower-bound* can be a valid string of the subtype, or left empty, which indicates that there is no lower bound. The value of *upper-bound* can be a valid string of the subtype, or null, which indicates that there is no upper bound.

Each bound value can be referenced using the quotation marks `""`. This is necessary if the bound value contains parentheses `()`, square brackets `[]`, commas `(,)`, quotation marks `""`, or backslashes `\`, because otherwise those characters will be considered part of the range syntax. To put a quotation mark or backslash in a bound value, put a backslash in front of it (and a pair of double quotation marks in a referenced bound value represents one quotation mark character, which is similar to the single quotation mark rule in SQL strings). In addition, you can avoid referencing and use backslash escapes to preventing data characters from being used as part of the return syntax. If you want to input a bound value that is an empty string, write `""`, indicating infinite bounds.

Spaces are allowed before and after a range value, but any space between parentheses() or square brackets[] is used as part of the upper or lower bound value (depending on the element type, the space may or may not represent a value).

Examples

Query all values between 3 and 7, including 3 and excluding 7:

```
SELECT '[3,7)>::int4range;  
int4range  
-----  
[3,7)  
(1 row)
```

Query all values between 3 and 7, excluding 3 and 7:

```
SELECT '(3,7)>::int4range;  
int4range  
-----  
[4,7)  
(1 row)
```

Query the value 4:

```
SELECT '[4,4]>::int4range;  
int4range  
-----  
[4,5)  
(1 row)
```

Query a range containing no values (normalized to **empty**):

```
SELECT '[4,4)>::int4range;  
int4range  
-----  
empty  
(1 row)
```

Constructing Range

Each range type has a constructor function with the same name. Using constructor functions is more convenient than writing a range literal constant because it avoids extra references to bound values. Constructor functions accept two or three parameters. Two parameters form a range in the standard form, where the lower bound is included and the upper bound is excluded. If a range contains three parameters, the third parameter specifies the range exclusion/inclusion type. The third parameter must be one of the following character strings: (), ([,], or []). The following is an example:

The complete form is: lower bound, upper bound, and textual parameters indicating the inclusion/exclusion of bounds.

```
SELECT numrange(1.0, 14.0, '[');  
numrange  
-----  
(1.0,14.0]  
(1 row)
```

If the third parameter is ignored, the range will be deemed as '[').

```
SELECT numrange(1.0, 14.0);  
numrange  
-----
```



```
[1.0,14.0)
(1 row)
```

Although '[' is specified here, it is converted to the standard form in the return result because `int8range` is a type of **Discrete Ranges**:

```
SELECT int8range(1, 14, '[');
int8range
-----
[2,15)
(1 row)
```

NULL indicates that the range is unbounded.

```
SELECT numrange(NULL, 2.2);
numrange
-----
(,2.2)
(1 row)
```

Discrete Ranges

A discrete range is a range in which its elements have a clearly defined "step", such as integer or date. When there is no valid value between two elements, they are adjacent.

Each element value in a discrete range has a next value and a previous value. You can use the next or previous value to convert between inclusion and exclusion. For example, in an integer range, [4,8] and (3,9) represent the same set of values, but this is not the case for ranges other than numeric ranges.

A discrete range should have a canonical function for identifying the step in the range. The normalization function can convert the equivalents of the range type to expressions of the same meanings, which are consistent with the inclusion or exclusion bounds. If no normalization function is specified, ranges with different formats are always considered unequal, even if they actually express the same set of values.

The built-in range types `int4range`, `int8range`, and `daterange` use the normalized form, which includes the lower bound and excludes the upper bound, that is, [). However, user-defined ranges types can use other conventions.

User-defined Range Types

Users can define range types. To create a range type subtype `float8`, execute the following commands:

```
CREATE TYPE floatrange AS RANGE (
    subtype = float8,
    subtype_diff = float8mi
);

SELECT '[1.234, 5.678]':floatrange;
```

Indexes

You can create GiST indexes for table columns of the range type. Example:

```
CREATE TABLE reservation (room int, during tsrange);
CREATE INDEX reservation_idx ON reservation USING gist (during);
```

GiST indexes can accelerate queries involving the following range operators: =, &&, <@, @>, <>, -|-, &<, and &. >. For details, see [Range Operators](#).

In addition, you can also create B-tree indexes on table columns of the range type. For these index types, basically the only useful range operation is equivalence. B-tree indexes for range types are primarily used to enable sorting within a query. They are not actually created.

21.5.20 Composite Types

A composite type represents the structure of a row or record, which is essentially a list of field names and their data types. DWS allows table columns to be declared as composite types. A composite type is essentially the same as the row type of a table. However, using [CREATE TYPE](#) avoids the need to create an actual table when only a type needs to be defined. A stand-alone composite type is useful as the parameter or return type of a function.

Declaration of Composite Types

DWS allows users to use [CREATE TYPE](#) to define composite types.

```
CREATE TYPE complex AS (  
    r    double precision,  
    i    double precision  
);  
  
CREATE TYPE inventory_item AS (  
    name      text,  
    supplier_id integer,  
    price     numeric  
);
```

After defining a composite type, you can create a table or function.

```
CREATE TABLE on_hand (  
    item    inventory_item,  
    count   integer  
);  
  
INSERT INTO on_hand VALUES (ROW('fuzzy dice', 42, 1.99), 1000);  
CREATE FUNCTION price_extension(inventory_item, integer) RETURNS numeric  
AS 'SELECT $1.price * $2' LANGUAGE SQL;  
  
SELECT price_extension(item, 10) FROM on_hand;
```

Constructing a Compound Value

To write a composite value as a literal constant, enclose the field value in parentheses and separate them with commas. You can add double quotation marks to any field value. This is mandatory if the field value contains commas or parentheses. The general format of a compound constant is as follows:

```
'( val1 , val2 , ... )'
```

'("fuzzy dice",42,1.99)' is a valid value of the **inventory_item** type.

To set a field to NULL, left the position empty. If you want a field to be an empty string, use quotation marks. In the following example, the first column is a non-null empty string, and the third column is NULL.

```
'( "",42, )'
```

ROW expressions can also be used to build composite values. Example:

```
ROW('fuzzy dice', 42, 1.99)
ROW("", 42, NULL)
```

Access Composite Types

To access a field in a composite column, you can write a dot and the name of the field, just like selecting a field from a table. To avoid confusion, you must use parentheses to distinguish it. For example, select some fields from table **on_hand**:

```
SELECT item.name FROM on_hand WHERE item.price > 9.99;
ERROR: missing FROM-clause entry for table "item"
```

This writing is confused with selecting fields from a table because **item** is regarded as a table name instead of a column name of **on_hand**. It must be written as follows:

```
SELECT (item).name FROM on_hand WHERE (item).price > 9.99;
```

If you need to use the table name (for example, in a multi-table query):

```
SELECT (on_hand.item).name FROM on_hand WHERE (on_hand.item).price > 9.99;
```

Objects with parentheses are now correctly interpreted as references to column **item**, from which fields can be selected.

Similar syntax applies when a field is selected from a compound value. For example, to select a field from the result of a function that returns a compound value, write as follows:

```
SELECT (my_func(...)).field FROM ...
```

If there are no additional parentheses, this generates a syntax error.

21.5.21 Data Types Supported by Column-Store Tables

[Table 21-27](#) lists the data types supported by column-store tables. Other data types that are not listed are not supported currently.

Table 21-27 Data Types Supported by Column-Store Tables

| Category | Data Type | Description | Length |
|--------------|-----------|--|-----------------|
| Numeric data | smallint | Small integer, also called INT2 | 2 |
| | integer | Typical choice for integer, also called INT4 | 4 |
| | bigint | Big integer, also called INT8 | 8 |
| | decimal | Arbitrary precision type | Variable length |
| | numeric | Arbitrary precision type | Variable length |

| Category | Data Type | Description | Length |
|----------------|----------------------------------|---|--|
| | real | Single-precision floating point | 4 |
| | double precision | Double-precision floating point | 8 |
| | smallserial | Two-byte auto-incrementing integer | 2 |
| | serial | Four-byte auto-incrementing integer | 4 |
| | bigserial | Eight-byte auto-incrementing integer | 8 |
| Money | money | Currency amount | 8 |
| Character data | character varying(n), varchar(n) | Variable-length string | Variable length |
| | character(n), char(n) | Fixed-length string | n |
| | character, char | Single-byte internal type | 1 |
| | text | Variable-length string | Variable length |
| | nvarchar2 | Variable-length string | Variable length |
| | clob | Big text object | Variable length |
| Date/Time | timestamp with time zone | Date and time (with time zone) | 8 |
| | timestamp without time zone | Date and time | 8 |
| | date | Date and time (Oracle compatibility mode); date (other compatibility modes) | When using Oracle compatibility mode, the storage space is 8 bytes, whereas in other compatibility modes, it is 4 bytes. |
| | time without time zone | Time within one day | 8 |

| Category | Data Type | Description | Length |
|---------------|---------------------|--|-----------------|
| | time with time zone | Time within one day (with time zone) | 12 |
| | interval | Time interval | 16 |
| JSON | json | - | - |
| | jsonb | - | - |
| RoaringBitmap | - | Efficient bitmap, which supports the bitmap calculation of datasets of the int type. | Variable length |

21.5.22 XML

XML data type stores Extensible Markup Language (XML) formatted data. XML data can also be stored as text, but the advantage of the XML data type is that it checks whether each stored value is a well-formed XML value. XML can store well-formed documents and content fragments defined by XML standards. A content fragment can have multiple top-level elements or character nodes.

For functions that support the XML data type, see [XML Functions](#).

Configuring XML Parameters

The syntax is as follows:

```
SET XML OPTION { DOCUMENT | CONTENT };  
SET xmloption TO { DOCUMENT | CONTENT };
```

If a string value is not converted to XML using the **XMLPARSE** or **XMLSERIALIZE** function, the **XML OPTION** session parameter determines it is **DOCUMENT** or **CONTENT**.

The default value is **CONTENT**, indicating that all types of XML data are allowed.

Example:

```
SET XML OPTION DOCUMENT;  
SET  
SET xmloption TO DOCUMENT;  
SET
```

Configuring Binary Data Encoding Format

Syntax:

```
SET xmlbinary TO { base64 | hex};
```

Example:

```
SET xmlbinary TO base64;  
SET
```

```
SELECT xmlelement(name foo, bytea 'bar');
xmlelement
-----
<foo>YmFy</foo>
(1 row)

SET xmlbinary TO hex;
SET

SELECT xmlelement(name foo, bytea 'bar');
xmlelement
-----
<foo>626172</foo>
(1 row)
```

Accessing XML Value

The XML data type is special, and it does not provide any comparison operators, because there is no general comparison algorithm for XML data, so you cannot retrieve data rows by comparing an XML value with a search value. An XML data entry is typically accompanied by an ID for retrieving. Alternatively, you can convert XML values into character strings. However, this is not widely applicable to common scenarios of XML value comparison.

21.6 Constant and Macro

Table 21-28 lists the constants and macros that can be used in DWS.

Table 21-28 Constants and macros

| Parameter | Description | Example SQL Statements |
|-----------------|--|--|
| CURRENT_CATALOG | Specifies the current database. | SELECT CURRENT_CATALOG; |
| CURRENT_ROLE | Current role | SELECT CURRENT_ROLE; |
| CURRENT_SCHEMA | Current database model | SELECT CURRENT_SCHEMA; |
| CURRENT_USER | Current user | SELECT CURRENT_USER; |
| LOCALTIMESTAMP | Current session time (without time zone) | SELECT LOCALTIMESTAMP; |
| NULL | This parameter is left blank. | - |
| SESSION_USER | Current system user | SELECT SESSION_USER; |
| SYSDATE | Current system date | SELECT SYSDATE;
or
SELECT now()::DATE; |

| Parameter | Description | Example SQL Statements |
|-----------|---|------------------------|
| USER | Current user, which is the same as the value of CURRENT_USER . | SELECT USER; |

21.7 Functions and Operators

21.7.1 String Processing Functions and Operators

21.7.1.1 Overview of String Processing Functions and Operators

The string processing functions and operators provided by DWS are used to obtain the string length as well as truncate, concatenate, convert, and compare data.

Before learning about the functions, you can learn about the concepts of bit, byte, and character.

- A bit is the basic unit of data in computers, representing either 0 or 1 in binary. These values correspond to the on/off state of a circuit or the positive/negative polarity of a magnetized surface.
- 1 byte = 8 bits. A byte the basic unit of data storage and transmission in computers.
- A character is a human-readable text symbol, such as a letter, digit, and punctuation. The database character encoding determines how a character is converted into bytes. For details about the database character encoding, see [CREATE DATABASE](#).

For example, in UTF-8 of DWS, an English character takes one byte, and a Chinese character takes three bytes. If a field is defined as VARCHAR(100), the maximum storage would be 100 characters x 3 bytes/character = 300 bytes. In GBK, a single Chinese character can take two bytes.

Application scenarios of string processing functions are as follows:

- **Obtaining the Length of a String**: Query the number of digits and byte length of a string. For example, query the number of digits of the string **World**.
- **Truncating a String**: Extract a substring from a long string. For example, extract **Hello** from **HelloWorld**.
- **Concatenating Strings**: Concatenate two strings, for example, concatenate **DATA** and **BASE** to obtain **DATABASE**.
- **Replacing and Padding Strings**: Replace a character or substring in a string with another character or substring, or delete redundant spaces. For example, delete the space of "**Database**".
- **Deleting a Specified Character**: Delete a specified character from a string, for example, delete **xx** from **DatxBasexx**.
- **Splitting a String**: Split a string into several parts as needed and separate the parts with delimiters. For example, split **hello world** into **hello**, **world**.

- **Converting a String**: Convert a string according to certain rules, for example, convert the case of letters.
- **Comparing Strings**: Compare two strings.
- **Obtaining the Position of a Character or Substring**: For example, obtain the second matching position of the substring **bcd** in the main string **abcdabcdabcd** starting from the second character.
- **Checking If a String Matches a Specific Pattern Using String Matching Functions**: For example, check whether **a** is in the regular expression of **[ac]**.

21.7.1.2 Obtaining the Length of a String

In DWS, you can use the following functions to obtain the length of a string, including the number of bits, bytes, and characters. For details about the differences among bits, bytes, and characters, see [Overview of String Processing Functions and Operators](#).

Table 21-29 Common functions

| Type | Description | Function | Example | Usage Difference |
|------|--|---------------------------|---|------------------|
| Bits | Obtain the number of digits in a string. | bit_length(string) | <pre>SELECT bit_length('world'); bit_length ----- 40 (1 row)</pre> | - |

| Type | Description | Function | Example | Usage Difference |
|------------|--|--|---|---|
| Bytes | Obtain the number of bytes in all string types. | octet_length(string) | <pre>SELECT octet_length('world'); octet_length ----- 5 (1 row)</pre> | <p>The functions of octet_length(string) and lengthb(string) are the same. When developing cross-platform databases such as MySQL and DWS, octet_length(string) is preferred. lengthb(string) is a mapping function for compatibility with Oracle and is used only for Oracle migration.</p> <p>lengthb(text/bpchar) is the same as lengthb(string), but its input parameter is of the text or bpchar type.</p> |
| | Obtain the number of bytes in all string types (only for compatibility with Oracle). | lengthb(string) | <pre>SELECT lengthb('abc'::text); lengthb ----- 3 (1 row)</pre> | |
| | Obtain the number of bytes in text and bpchar (only for compatibility with Oracle). | lengthb(text/bpchar) | <pre>SELECT lengthb('abc'::CHAR(5)); lengthb ----- 5 (1 row)</pre> | |
| Characters | Obtain the number of characters in a string. | char_length(string) or character_length(string)
length(string) | <pre>SELECT length('database'); length ----- 8 (1 row)</pre> | <p>length(string) is equivalent to char_length(string), but its format is simpler.</p> |
| | Obtain the number of characters in a string in the specified encoding format. | length(string bytea, encoding name) | - | - |

bit_length(string)

Description: Obtains the number of bits in a string. That is, the number of bytes multiplied by 8. The number of bits in a string depends on the database encoding

mode. For example, in GBK, a Chinese character is 2 bytes. In UTF-8 and SQL_ASCII, a Chinese character is 3 bytes.

Return type: integer

Example:

```
SELECT bit_length('world');
bit_length
-----
         40
(1 row)
```

octet_length(string)

Description: Obtains the number of bytes in a string. It is important for processing multi-byte characters (such as UTF-8 characters and special symbols) and can accurately obtain the storage size of a string.

Return type: integer

Example:

```
SELECT octet_length('data');
octet_length
-----
          4
(1 row)
```

lengthb(string)

Description: Obtains the length of a string in bytes. It shares the same functions as [octet_length\(string\)](#) and is only used for compatibility with Oracle. If data is migrated from Oracle, use **lengthb(string)**. However, if workloads need to run across databases (for example, DWS and MySQL), use **octet_length(string)** because MySQL does not have the **lengthb** function.

The return result depends on character sets (GBK and UTF-8).

Return type: integer

Example:

```
SELECT lengthb('hello');
lengthb
-----
        5
(1 row)
```

lengthb(text/bpchar)

Description: Obtains the number of bytes of a specified string. The input parameter can be text or bpchar (equivalent to **CHAR(n)**).

- **lengthb(text)** indicates the byte length of actual characters (without padding spaces), which is suitable for most scenarios.
- **lengthb(bpchar)** indicates the byte length of a fixed-length string (including padding spaces), which is suitable for strict fixed-length scenarios (such as

fixed-length codes and certificate numbers). It is not recommended for daily use (implicit problems may be caused by space padding).

Return type: integer

NOTE

- For a string containing newline characters, for example, a string consisting of a newline character and a space, the value of **length** and **lengthb** in DWS is 2.
- This function returns the number of bytes in a specified string. For multi-byte character encoding, such as UTF-8, one character may occupy multiple bytes.

Example:

```
SELECT lengthb('hello');
lengthb
-----
      5
(1 row)
```

Examples of input parameters of the text and bpchar types:

```
-- Case 1: Specify the input parameters as text.
SELECT lengthb('abc'::text); -- 'abc' occupies 3 bytes (UTF-8), and the returned value is 3.
-- Case 2: Specify the input parameters as bpchar or CHAR(5).
SELECT lengthb('abc'::CHAR(5)); -- The value is stored as 'abc..', and the returned value is 5 (each space occupies 1 byte).
```

```
utf8=> SELECT lengthb('abc'::text);
lengthb
-----
      3
(1 row)

utf8=> SELECT lengthb('abc'::CHAR(5));
lengthb
-----
      5
(1 row)
```

char_length(string) or character_length(string)

Description: Obtains the number of characters in a string.

Return type: integer

NOTE

For multi-byte character encoding (such as UTF-8), one character may occupy multiple bytes.

Example:

```
SELECT char_length('hello');
char_length
-----
      5
(1 row)
```

length(string)

Description: Obtains the character length (number of characters) of a string. It is equivalent to `char_length(string)` and is simpler.

Return type: integer

Example:

```
SELECT length('database');
length
-----
      8
(1 row)
```

length(string bytea, encoding name)

Description: Obtains the number of characters in a string in the specified encoding format. In the specified encoding format, **string** must be valid.

Return type: integer

Example:

```
SELECT length('database', 'UTF8');
length
-----
      8
(1 row)
```

21.7.1.3 Truncating a String

Truncation strings is a key technical capability in multidimensional analysis of data warehouses.

Typical applications:

- **Extracting core information:** For example, extract the core serial number **0129A** from **EC2023-BJ-0129A**, and use **substring()** to capture the administrative division code and birth date in a 18-digit ID card code.
- **Masking data for privacy protection:** For example, use **right()** to extract the last four digits of a mobile number for data masking
- **Extracting information accurately:** For example, use **regexp_matches** to match multiple character strings starting with the letter **b**.
- **Clearing redundant data:** For example, clear the redundant spaces or null characters before and after an order No.

DWS provides multiple functions for you to extract strings.

Table 21-30 Common functions

| Type | Description | Function | Example | Usage Difference |
|---------------------------------|---|---|---|---|
| Remove characters at both ends. | Remove the specified characters or spaces at both ends of a string. | btrim(string text [, characters text]) | <pre>SELECT btrim(' string '); btrim ----- string (1 row)</pre> | - |
| Extract a substring, in bytes. | Extract a substring from a given string, where the starting position and length are specified in bytes. | substrb(string,from,count) | <pre>SELECT substrb('ABCDEF',2,2); substrb ----- BC (1 row)</pre> | These four functions work similarly but differ in two ways: what they extract (either the full remaining parts or just specific middle bytes) and their input parameters (whether they accept text-only or all string types). |
| | Extract a substring from a given string, where the starting position and length are specified in bytes. It and the above function work similarly, but the difference is that the input parameter type of the function is text | substrb(text,int,int) | - | |

| Type | Description | Function | Example | Usage Difference |
|------|---|-----------------------------|--|------------------|
| | Extract a substring from a given string, where the starting position is specified in bytes. If from is a negative number, the starting position is counted backward from the end of the string. (It means that the last <i>n</i> bytes are extracted.) | substrb(string,from) | <pre>SELECT substrb('ABCDEF',-2); substrb ----- EF (1 row)</pre> | |
| | Extract a substring from a given string, where the starting position is specified in bytes. They work similarly, but the difference is that the input parameter type of the function is text. | substrb(text,int) | <pre>SELECT substrb('string',2); substrb ----- tring (1 row)</pre> | |

| Type | Description | Function | Example | Usage Difference |
|-------------------------------------|--|---|--|---|
| Extract a substring, in characters. | Extract a substring from a given string, where the starting position and length are specified in characters. | substr(string,from,count) | <pre>SELECT substr('database',0,4) ; substr ----- data (1 row)</pre> | substr(string,from,count) and substring(string [from int] [for int]) work similarly but differ in two ways: what they extract (either the full remaining parts or just specific middle characters) and their SQL syntax (the former is simpler and the latter is standard SQL). |
| | Extract a substring from a given string, where the starting position is specified in characters. | substr(string,from) | <pre>SELECT substr('database',5); substr ----- base (1 row)</pre> | |
| | Extract a substring from a given string, where the starting position and length are specified in characters. It and substr(string,from,count) work similarly. | substring(string [from int] [for int]) | - | |
| | Extract the first several characters. | left(string, n int) | - | - |
| | Extract the last several characters. | right(string, n int) | - | - |

| Type | Description | Function | Example | Usage Difference |
|---|--|--|---|---|
| Extract substrings by regular expression pattern. | Extract a portion of a string that matches a specified regular expression pattern. | substring(string from pattern) | Click the link on the left to view the details. | regexp_substr and substring are similar. substring is a standard SQL function, while regexp_substr is an extended function with stronger regular expression capabilities.
regexp_matches returns text arrays, which is suitable for extracting group information. |
| | Extract a portion of a string that matches a specified regular expression pattern. The escape character can be set. | substring(string from pattern for escape) | | |
| | It is similar to the preceding function, but has a strong regular expression capability. | regexp_substr(text, text) | | |
| | It is similar to the preceding function, but has a strong regular expression capability. | regexp_substr(source_char, pattern) | | |
| | Extract substrings and return text arrays. It works well when you need to match multiple conditions or extract group data. | regexp_matches(string text, pattern text [, flags text]) | | |

btrim(string text [, characters text])

Description: Removes the specified characters from both ends of a string. Only the characters at both ends are processed. The characters in the middle will not be removed even if they match.

Return type: text

Parameters:

- **string**: (mandatory) target string.
- **characters**: (optional) characters to be removed from both ends of the target string. If it is not specified, all whitespace characters are removed by default.

Example:

```
SELECT btrim('sring' , 'ing');
btrim
-----
sr
(1 row)

SELECT btrim(' sting '); -- If characters are not specified, all spaces are deleted by default.
btrim
-----
sting
(1 row)

SELECT btrim('xxdataxxbasex','x'); -- Only the x at both ends is deleted. The x in the middle is not deleted.
btrim
-----
dataxxbase
(1 row)
```

substrb(string,from,count)

Description: Truncates a substring of a specified length from a string. **from** specifies the position where the extraction starts. **count** specifies the length of the extracted substring.

It and [substr\(string,from,count\)](#) work similarly, but its calculation unit is byte.

- If **from** starts at 0, it is typically treated as 1.
- If **from** is a positive value, extract **count** bytes starting from **from**.
- If **from** is a negative value, extract the last **n count** bytes in the string, in which **n** indicates the absolute value of **from**.
- When **count** is less than 1, null is returned.

Return type: bytea

Example:

```
SELECT substrb('ABCDEF',2,2);
substrb
-----
BC
(1 row)
```

substrb(text,int,int)

Description: Extracts substrings from a string. The first **int** indicates the start byte position, and the second **int** indicates the number of bytes to be extracted.

Return type: text

Example:

```
SELECT substrb('string',2,3); ---Extracts three bytes starting from the second byte t.
substrb
-----
tri
(1 row)
```

substrb(string,from)

Description: Truncates a specified substring from a string. **from** indicates the starting position. It and **substr(string,from)** work similarly, but its calculation unit is byte.

- If **from** starts at 0, it is typically treated as 1.
- If **from** is a positive value, all bytes from **from** to the end are extracted.
- If **from** is a negative value, the last *n* bytes in the string are extracted, in which *n* indicates the absolute value of **from**.

Return type: bytea

Example:

```
SELECT substrb('ABCDEF',-2);
substrb
-----
EF
(1 row)
```

substrb(text,int)

Description: Extracts a substring from a string. **int** indicates the start byte position of the subtraction.

Return type: text

Example:

```
SELECT substrb('string',2);
substrb
-----
tring
(1 row)
```

substr(string,from,count)

Description: Truncates a substring of a specified length from a string.

from specifies the position where the extraction starts. **count** specifies the length of the extracted substring.

- If **from** starts at 0, it is typically treated as 1.
- If **from** is a positive value, extract **count** characters starting from **from**.
- If **from** is a negative value, extract the last **n count** characters in the string, in which **n** indicates the absolute value of **from**.
- When **count** is less than 1, null is returned.

Return type: varchar

Example:

When **from** is 0, extract the first four characters from the string **database**.

```
SELECT substr('database',0,4);
substr
-----
data
(1 row)
```

When **from** is a positive number, extract the first four characters from the fifth character of the string **database**.

```
SELECT substr('database',5,4);
substr
-----
base
(1 row)
```

If **from** is a negative number, truncate three characters starting from the last fourth character of the string **database**.

```
SELECT substr('database',-4,3);
substr
-----
bas
(1 row)
```

substr(string,from)

Description: Truncates a specified substring from a string.

from indicates the start position of the substring.

- If **from** starts at 0, it is typically treated as 1.
- If **from** is a positive value, all characters from **from** to the end are extracted.
- If **from** is a negative value, the last *n* characters in the string are extracted, in which *n* indicates the absolute value of **from**.

Return type: varchar

Example:

When **from** starts at 0, extract all characters from the first character to the end of the string **database**.

```
SELECT substr('database',0);
substr
-----
database
(1 row)
```

When **from** is a positive value, extract all characters from the fifth character to the end of the string **database**.

```
SELECT substr('database',5);
substr
-----
base
(1 row)
```

When **from** is a negative value, extract all characters from the fourth character from the end to the end of the string **database**.

```
SELECT substr('database',-4);
substr
-----
base
(1 row)
```

substring(string [from int] [for int])

Description: Extracts a substring. **from int** indicates the start position of the truncation. **for int** indicates the number of characters truncated. It and [substr\(string,from,count\)](#) work similarly.

Return type: text

Example:

Extract three characters starting from the second character **h** in the string **Thomas**, that is, **hom**.

```
SELECT substring('Thomas' from 2 for 3);
substring
-----
hom
(1 row)
```

left(str text, n int)

Description: Returns first **n** characters in a string.

- In the ORA- or TD-compatible mode, all but the last **|n|** characters are returned if **n** is a negative value.
- In the MySQL-compatible mode, an empty string is returned if **n** is a negative value.

Return type: text

Example:

```
SELECT left('database', 4);
left
-----
data
(1 row)
```

right(str text, n int)

Description: Returns the last **n** characters in a string.

- In the ORA- or TD-compatible mode, all but the first **|n|** characters are returned if **n** is a negative value.
- In the MySQL-compatible mode, an empty string is returned if **n** is a negative value.

Return type: text

Example:

```
SELECT right('abcde', 2);
right
-----
de
(1 row)

SELECT right('abcde', -2);
right
-----
cde
(1 row)
```

substring(string from *pattern*)

Description: Extracts a substring from a string that matches a specified pattern, which is a POSIX regular expression. **string** indicates the original string, and **pattern** indicates the pattern to be matched. If no match is found, NULL is returned. If a match is found, the matched part is returned.

Return type: text

Example:

```
SELECT substring('Thomas' from '...$'); --- Match the last three consecutive characters.
substring
-----
mas
(1 row)

SELECT substring('Thomas' from '^...'); --- Match the first three consecutive characters.
substring
-----
Tho
(1 row)

SELECT substring('Tom' from '....$'); --- Match the last four consecutive characters. Tom has only three
characters and cannot match the condition. Null is returned.
substring
-----
(1 row)

SELECT substring('order_20251120_12345' from '_[0-9]+_'); --- Obtain the order No. between two
underscores (_).
substring
-----
20251120
(1 row)

SELECT substring('tel:13812345678,status:valid' from 'tel:([0-9]{11})'); --- Extract the mobile number
information.
substring
-----
13812345678
(1 row)

SELECT substring('foobar' from 'o(.)b'); -- Match the o(.)b rule. o indicates that the first letter o is matched,
(.) indicates that any character is matched and captured, and b indicates that the letter b is matched. In
foobar, oob is the matched substring. (.) is the middle character (o) of the substring.
substring
-----
```

```
o
(1 row)

SELECT substr('foobar' from '(o(.b))'); -- Use nested capture rules. The outer group captures oob, while
the inner group captures the middle o. The function returns only the first captured group, so the result is
oob.
substr
-----
oob
(1 row)
```

 NOTE

If the POSIX pattern contains any parentheses, the portion of the text that matched the first parenthesized sub-expression (the one whose left parenthesis comes first) is returned. If you want to use parentheses in an expression without causing this exception, you can add a pair of parentheses outside the entire expression.

substring(string from *pattern* for *escape*)

Description: Extracts the substring that matches the SQL regular expression. **string** indicates the original string, **pattern** indicates the pattern that the string must match, and **escape** indicates the escape character, which is specified by **for**. If no pattern is matched, null is returned.

To identify the specific portion of the pattern that should be returned upon a successful match, the pattern must contain two occurrences of the designated escape character, each followed by a double quotation mark ("). The text in the data string that matches the part of the pattern between these two markers is the substring returned by the function.

Return type: text

Example:

```
SELECT substr('Thomas' from '%#"o_a#"_' for '#');
substr
-----
oma
(1 row)
```

Example explanations:

1. **%#"o_a#"_** is the pattern, and **#** is the escape character.
2. **%** indicates the wildcard that matches any number of any prefix. In this example, the prefix **Tho** is matched.
3. **#"** is a marker used to match the text between two **#"**, that is, **o_a**.
4. The last underscore (**_**) indicates that the suffix character **s** is matched.

In conclusion, the pattern body **%#"o_a#"_** is expanded to **% o_a _**, which means that the pattern body matches "any prefix" + o + "single any character" + a + "single any character".

If the string Thomas needs to match this pattern, that is, only the text between the two number signs (**#**) is extracted, that is, **o_a**, where **_** can indicate any character. The result is **oma**.

Example:

Extract the four digits after FX. (The express delivery number is FX**5892**20251124, which is in the format of FX + 4-digit number + 8-digit timestamp.)

```
SELECT substring('FX589220251124' from 'FX#"____#"%' for '#');
substring
-----
5892
(1 row)

SELECT substring('FX564320241024' from 'FX#"____#"%' for '#');
substring
-----
5643
(1 row)
```

regexp_substr(text,text)

Description: Extracts substrings from a regular expression. The first **text** indicates the original string to be processed, and the second **text** indicates the defined regular expression. It is similar to **substr**. When a regular expression contains multiple parallel brackets, all of them are processed.

Return type: text

Example:

```
SELECT regexp_substr('str','[ac]'); --- Match any character in ac from str and return null.
regexp_substr
-----
(1 row)
```

Example: Obtain the first 10 digits from the express number. The other digits after the first 10 digits are timestamps and can be ignored.

```
SELECT regexp_substr('FX58920235822025112404', '\d{10}'); --Search for the first occurrence of 10
consecutive digits in the string by scanning it from left to right.
regexp_substr
-----
5892023582
(1 row)
```

regexp_substr(source_char, pattern)

Description: Extracts substrings from a regular expression. **source_char** indicates the original string, and **pattern** indicates the defined regular expression.

Return type: varchar

Example:

```
-- Find the part of the string that starts with a comma, has at least one non-comma character, and ends
with another comma.
SELECT regexp_substr('500 Hello World, Redwood Shores, CA', '^[^,]+,') "REGEXPR_SUBSTR";
REGEXPR_SUBSTR
-----
, Redwood Shores,
(1 row)
```

regexp_matches(string text, pattern text [, flags text])

Description: Returns all substrings that match the POSIX regular expression. The returned result is a text array. If the pattern does not match, the function returns no rows. This function is flexible in processing complex text parsing, especially when you need to match multiple conditions or extract group data.

- **string** indicates the original string.
- **pattern** indicates the pattern the string must match. The pattern can contain multiple sub-expressions in parentheses, for example, **(bar)(beque)**. If the pattern is matched, a text array with multiple elements is returned, for example, **{bar,beque}**. If the pattern does not contain sub-expressions in parentheses, a text array with a single element is returned, for example, **{barbeque}**.
- **flags** is an optional parameter. If **flags** is set to **i**, the pattern matching is case insensitive. If **flags** is set to **g**, it indicates that the function finds all matches of the pattern within the string, not just the first one.

NOTICE

If the last parameter **flags** is provided but the parameter value is an empty string (") and the SQL compatibility mode of the database is set to ORA, the returned result is null. To resolve this problem, you can:

- Change the database SQL compatibility mode to TD.
- Do not provide the last parameter or do not set the last parameter to an empty string.

Return type: multiple lines of text as an array (setof text[]).

```
-- The rule has two parts: bar and beque. The result is a text array containing two elements.
```

```
SELECT regexp_matches('foobarbequebaz', '(bar)(beque)');
regexp_matches
```

```
-----
{bar,beque}
(1 row)
```

```
-- The rule does not contain parentheses. The result is a text array that contains only one element.
```

```
SELECT regexp_matches('foobarbequebaz', 'barbeque');
regexp_matches
```

```
-----
{barbeque}
(1 row)
```

```
-- The rule matches strings starting with 'b' followed by one or more non-'b' characters. It splits the match into two groups: bar and beque. Since g is not included, only the first line matches.
```

```
SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)');
regexp_matches
```

```
-----
{bar,beque}
(1 rows)
```

```
-- Compared with the preceding example, if g is provided, the matching continues and the second line of the text array is displayed.
```

```
SELECT regexp_matches('foobarbequebazilbarfbonk', '(b[^b]+)(b[^b]+)', 'g');
regexp_matches
```

```
-----
{bar,beque}
{bazil,barf}
(2 rows)
```

```
-- The rule extracts all 11-digit mobile numbers from the text and causes the function to find all matches of the pattern within the string. All lines are returned.
```

```
SELECT regexp_matches('13812345678,13987654321', '\d{11}', 'g');
regexp_matches
```



```
{13812345678}
{13987654321}
(2 rows)

-- The rule adds i to make the pattern matching case-insensitive.
SELECT regexp_matches('fooBarBeQuebaz', 'barbeque', 'i');
regexp_matches
-----
{BarBeQue}
(1 row)

-- The rule does not contain i, so the pattern matching is case sensitive and null is returned.
SELECT regexp_matches('fooBarBeQuebaz', 'barbeque');
regexp_matches
-----
(0 rows)
```

CAUTION

When a subquery is absent, and the **regexp_matches** function fails to find a match, the table data remains hidden, which is contrary to what is usually intended. To avoid this issue, employ the **regexp_substr(text,text)** function, which can deliver the expected results even without a subquery.

```
SELECT * FROM tab;
c1 | c2
---+---
dws | dws
(1 row)

SELECT c1, regexp_matches(c2, '(bar)(beque)') FROM tab;
c1 | regexp_matches
---+-----
(0 rows)

SELECT c1, c2, (SELECT regexp_matches(c2, '(bar)(beque)')) FROM tab;
c1 | c2 | regexp_matches
---+---+-----
dws | dws |
(1 row)
```

21.7.1.4 Concatenating Strings

rawcat(raw,row)

Description: Concatenates the given strings.

Return type: raw

Example:

```
SELECT rawcat('ab','cd');
rawcat
-----
ABCD
(1 row)
```

repeat(string text, number int)

Description: Outputs a string repeatedly for a specified number of times. If **number int** is set to zero or a negative value, an empty string is returned.

Return type: text

Example:

```
SELECT repeat('abc', 4);
repeat
-----
abcbabcbabcb
(1 row)
```

string || string

Description: Concatenates strings.

Return type: text

Example:

```
SELECT 'DA' || 'TABASE' AS RESULT;
result
-----
DATABASE
(1 row)
```

string || non-string or non-string || string

Description: Concatenates strings and non-strings.

Return type: text

Example:

```
SELECT 'Value: ' || 123 AS RESULT;
result
-----
Value: 123
(1 row)
```

concat(str1,str2)

Description: Connects **str1** and **str2** and returns the string.

- In the ORA- or TD-compatible mode, a combination of all the non-null strings is returned.
- In the MySQL-compatible mode, **NULL** is returned if an input string is **NULL**.

Return type: varchar

Example:

```
SELECT concat('Hello', ' World!');
concat
-----
Hello World!
(1 row)
```

concat_ws(sep text, str"any" [, str"any" [, ...]])

Description: Concatenates all but the first parameter with separators. The first parameter is used as the separator string.

Return type: text

Example:

```
SELECT concat_ws(',', 'ABCDE', 2, NULL, 22);
concat_ws
-----
ABCDE,2,22
(1 row)
```

21.7.1.5 Replacing and Padding Strings

lpad(string text, length int [, fill text])

Description: Fills up the string to length by appending the characters fill (a space by default). If the string is already longer than length, it is truncated (on the right).

Return type: text

Example:

```
SELECT lpad('data', 8, 'xy');
lpad
-----
xyxydata
(1 row)
```

overlay(string placing string FROM int [for int])

Description: Replaces the part of a string starting from a specified position with a specified string. It can flexibly replace substring.

Parameters:

FROM int: starts the replacement from the *n*th character of the first string.

for int: (optional) number of characters to be replaced in the first string. If this parameter is not specified, all characters starting from **from** are replaced by default.

Return type: text

Example: Replace part of **abcdef** starting at position 3 with **xyz**. The result is **abxyzf**.

```
SELECT overlay('abcdef' placing 'xyz' FROM 3);
overlay
-----
abxyzf
(1 row)
```

Replace three characters starting at position 2 in **hello** with **world**. The result is **hworldo**.

```
SELECT overlay('hello' placing 'world' from 2 for 3 );
overlay
-----
hworldo
(1 row)
```

replace(string text, from text, to text)

Description: Replace all occurrences in string of substring **from** with substring **to**.

NOTE

The case is sensitive, so you need to ensure that the case of **from text** and **to text** matches that of the original string.

Return type: text

Example:

```
SELECT replace('databasedata', 'data', 'xxx');
replace
-----
xxxbasexxx
(1 row)
```

rpad(string text, length int [, fill text])

Description: Fills up **string** to **length** by appending the characters **fill**. If the string is longer than the specified length, the string is truncated on the right.

Parameters:

- **string**: original string to be processed.
- **length**: length of the target string.
- **fill** (optional): character or string used for filling. The default value is a space.

Return type: text

Example:

Fill the original string (**data**) of length 4 to a target length of 10 using the fill characters **xy**.

```
SELECT rpad('data', 10, 'xy');
rpad
-----
dataxyxyxy
(1 row)
```

Fill the original string **database** using the fill characters **xy**. The string **database** is longer than the target length **4**, and the string is truncated from the end. The returned string is **data**.

```
SELECT rpad('database',4,'xy');
rpad
-----
data
(1 row)
```

rtrim(string text [, characters text])

Description: Removes the specified characters from the right (end) of a string.

Parameters:

- **string**: original string to be processed.
- **characters** (optional): set of characters to be removed from the right of the string. By default, all blank characters (for example, spaces) are removed.

Return type: text

Example:

If the parameter **characters** is not specified, all blank characters on the right of the string **Database** are removed by default.

```
SELECT rtrim('Database ');
rtrim
-----
Database
(1 row)
```

Remove the specified character **x** from the string.

```
SELECT rtrim('trimxxx', 'x');
rtrim
-----
trim
(1 row)
```

rpad(string varchar, length int [, fill varchar])

Description: Adds specified characters to the right of a string until the string reaches the specified length. If the string is longer than the specified length, the string is truncated on the right.

Parameters:

- **string**: original string to be processed.
- **length**: length of the target string. **length** in DWS indicates the character length. One Chinese character is counted as one character.
- **fill** (optional): character or string used for filling. The default value is a space.

Return type: varchar

Example:

```
SELECT rpad('data',10,'xyz');
rpad
-----
dataxyzxyz
(1 row)
SELECT rpad('hi',5,'abcdefg');
rpad
-----
hiabc
(1 row)
```

replace(string varchar, search_string varchar, replacement_string varchar)

Description: Searches for a specific substring (**search_string**) in a string and replaces it with another string (**replacement_string**).

Parameters:

- **string**: original string to be processed.
- **search_string**: substring to be replaced.
- **replacement_string**: new substring used to replace **search_string**.

Return type: varchar

Example:

Replace all substrings **ab** in the string **ababab and mabab** with **cd**.

```
SELECT replace('ababab and mabab','ab','cd');
      replace
-----
cdcdcd and mcdcd
(1 row)
```

lpad(string varchar, length int[, repeat_string varchar])

Description: Adds specified characters to the left of a string until the string reaches the specified length. If the length of the original string is greater than the target length (**length parameter**), the target string is truncated from the right to a string of the **length**.

Parameters:

- **string**: original string to be padded.
- **length**: length of the target string.
- **repeat_string**: (optional) characters used for padding. By default, spaces are used for padding.

Return type: varchar

Example:

Set original string length (**base**) to **4** and the target length to **8**, and use asterisks (*) to pad four characters. After the padding, the string becomes ******base**.

```
SELECT lpad('base',8,'*');
      lpad
-----
****base
(1 row)
```

Set original string length (**database**) to **8** and the target length to **6**, and truncate the string **database** from the right to a string of 6 characters. **databa** is returned.

```
SELECT lpad('database',6,'data');
      lpad
-----
databa
(1 row)
```

regexp_replace(string, pattern, replacement [,flags])

Description: Replaces substring matching POSIX regular expression. The source string is returned unchanged if there is no match to the pattern. If a match is found, the matching substring in the returned string is replaced with the **replacement** string.

The **replacement** string can contain **\n**, where **\n** ranges from 1 to 9, indicating that the substring of the *n*th parenthesized subexpression in the matching pattern

within the string should be inserted. Additionally, it can contain `\&`, indicating that the substring matching the entire pattern should be inserted.

flags (optional parameter) contains zero or multiple single-letter flags that change function behavior. The following table lists the options of **flags**.

Table 21-31 Options of the flags parameter

| Option | Description |
|--------|--|
| g | Replaces all the matched substrings. (By default, only the first matched substring is replaced.) |
| B | By default, uses the Henry Spencer regex library and its regex syntax. Specifying the B option prioritizes using the Boost.Regex library and its regex syntax.

In the following cases, the Henry Spencer's regular expression library and its regular expression syntax will be used even if this option is specified: <ul style="list-style-type: none">Any combination of p, q, w, and x characters is included in the flags.The string or pattern parameters contain multibyte characters. |
| b | Indicates POSIX Basic Regular Expressions (BREs) for matching. |
| c | Indicates the case-sensitive matching. |
| e | Indicates POSIX Extended Regular Expressions (EREs) for matching. When neither b nor e is specified, if the Henry Spencer regex library is chosen, it follows the Advanced Regular Expressions (AREs), similar to Perl Compatible Regular Expressions (PCREs). If the Boost.Regex library is chosen, it adheres to PCREs. |
| i | Indicates the case-insensitive matching. |
| m | Indicates the newline-sensitive matching, synonymous with the n option. |
| n | Indicates the newline-sensitive matching. When this option is active, the newline character influences the matching of metacharacters (<code>.</code> , <code>^</code> , <code>\$</code> , and <code>[]</code>). |
| p | Indicates the partially newline-sensitive matching. When this option is active, the newline character influences the matching of metacharacters (<code>.</code> and <code>[]</code>), partially compared to the n option. |
| q | Resets the regular expression to a double-quoted text string, treating all characters as literals. |
| s | Indicates the non-newline-sensitive matching. |
| t | Indicates the tight syntax (default). When this option takes effect, all characters matter. |
| w | Indicates the inverse partially newline-sensitive matching. When this option is active, the newline character influences the matching of metacharacters (<code>^</code> and <code>\$</code>), inversely partially compared to the n option. |

| Option | Description |
|--------|--|
| x | Indicates the expanded syntax. In contrast to the tight syntax, whitespace characters in regular expressions are ignored in the expanded syntax. Whitespace characters include spaces, horizontal tabs, newlines, and any characters belonging to the space character set. |

Return type: varchar

Example:

```
SELECT regexp_replace('Thomas', '[mN]a.', 'M');
regexp_replace
-----
ThM
(1 row)
SELECT regexp_replace('foobarbaz', 'b(..)', E'X\\1Y', 'g') AS RESULT;
result
-----
fooXarYXazY
(1 row)
```

SPACE(n int)

Description: Returns a string consisting of **n** spaces. If the parameter contains NULL, NULL will be returned.

Return type: text

Example:

```
SELECT SPACE(2) as result;
result
-----
(1 row)
```

21.7.1.6 Deleting a Specified Character

trim([leading |trailing |both] [characters] from string)

Description: Deletes specific characters from the beginning, end, or both ends of a string.

Parameters:

- **leading**: deletes spaces or specified characters at the beginning of a string.
- **trailing**: deletes spaces or specified characters at the end of a string.
- **both**: deletes spaces or specified characters at the beginning and end of a string (default behavior).
- **characters**: (optional) specifies specific characters to be deleted. If this parameter is not specified, spaces are deleted by default.
- **string**: original string to be processed.

Return type: varchar

Examples:

Delete the characters **x** at the beginning and end of the string **xTomxx**.

```
SELECT trim(BOTH 'x' FROM 'xTomxx');
      btrim
-----
      Tom
(1 row)

SELECT trim('x' FROM 'xTomxx');
      btrim
-----
      Tom
(1 row)
```

Delete the character **x** at the beginning of the string **xTomxx**.

```
SELECT trim(LEADING 'x' FROM 'xTomxx');
      ltrim
-----
      Tomxx
(1 row)
```

Delete the character **x** at the end of the string **xTomxx**.

```
SELECT trim(TRAILING 'x' FROM 'xTomxx');
      rtrim
-----
      xTom
(1 row)
```

rtrim(string [, characters])

Description: Deletes the specified characters from the end of a string.

Parameters:

- **string**: original string to be processed.
- **characters** (optional): specifies the set of characters to be deleted from the end of a string. By default, all blank characters (for example, spaces) are deleted.

Return type: varchar

Example:

```
SELECT rtrim('DataxBasexx','x');
      rtrim
-----
      DataBase
(1 row)

SELECT rtrim(' Data ');
      rtrim
-----
      Data
(1 row)
```

ltrim(string [, characters])

Description: Deletes the specified strings from the beginning of a string.

Parameters:

- **string**: original string to be processed.
- **characters** (optional): specifies the set of characters to be deleted from the beginning of a string. By default, all blank characters (for example, spaces) are deleted.

Return type: varchar

Example:

```
SELECT ltrim('xxDataxBaSe','x');
ltrim
-----
DataxBaSe
(1 row)

SELECT ltrim('  Data ');
ltrim
-----
Data
(1 row)
```

21.7.1.7 Splitting a String

regexp_split_to_array(string text, pattern text [, flags text])

Description: Splits **string** using a POSIX regular expression as the delimiter. It is similar to **regexp_split_to_table**, but it returns the result as a text array.

Return type: text[]

Example:

```
SELECT regexp_split_to_array('hello world', E'\\s+');
regexp_split_to_array
-----
{hello,world}
(1 row)
```

regexp_split_to_table(string text, pattern text [, flags text])

Description: Splits **string** using a POSIX regular expression as the delimiter. If there is no match to the pattern, the function returns the string. If there is at least one match, for each match it returns the text from the end of the last match (or the beginning of the string) to the beginning of the match. When there are no more matches, it returns the text from the end of the last match to the end of the string.

The **flags** parameter includes zero or more single-character flags that alter the function's behavior. Flag **i** specifies case-insensitive matching, while flag **g** specifies replacement of each matching substring rather than only the first one.

Return type: setof text

Example:

```
SELECT regexp_split_to_table('hello world', E'\\s+');
regexp_split_to_table
-----
hello
world
(2 rows)
```

 CAUTION

If there is no subquery and the **regexp_split_to_table** function does not find a match, the data in the table will not be output. This is generally not the desired result and such usage should be avoided.

```
SELECT * FROM tab;
c1 | c2
-----+-----
dws |
(1 row)

SELECT c1, regexp_split_to_table(c2, E'\\s+') FROM tab;
c1 | regexp_split_to_table
-----+-----
(0 rows)

SELECT c1, (select regexp_split_to_table(c2, E'\\s+')) FROM tab;
c1 | regexp_split_to_table
-----+-----
dws |
(1 row)
```

split_part(string text, delimiter text, field int)

Description: Splits the original string into several parts based on the specified delimiter (**delimiter** parameter) and returns the given field (**field** parameter). This function is used to extract specific fields from composite strings.

Parameters:

- **string**: original string to be split.
- **delimiter**: delimiter used to split strings.
- **field**: given field returned after the split. The value starts from 1.

Return type: text

Example:

Split the string **abc~@~def~@~ghi** by **~@~** to obtain three parts (**abc**, **def**, and **ghi**), and return the second part, that is, **def**.

```
SELECT split_part('abc~@~def~@~ghi', '~@~', 2);
split_part
-----
def
(1 row)
```

21.7.1.8 Converting a String

upper(string)

Description: Converts the string into the uppercase.

Return type: varchar

Example:

```
SELECT upper('database');
upper
```

```
-----  
DATABASE  
(1 row)
```

ucase(string)

Description: Converts the string into the uppercase.

Return type: varchar

Example:

```
SELECT ucase('database');  
ucase  
-----  
DATABASE  
(1 row)
```

lower(string)

Description: Converts the string into the lowercase.

Return type: varchar

Example:

```
SELECT lower('DATABASE');  
lower  
-----  
database  
(1 row)
```

lcase(string)

Description: Converts the string into the lowercase.

Return type: varchar

Example:

```
SELECT lcase('DATABASE');  
lcase  
-----  
database  
(1 row)
```

initcap(string)

Description: Converts the first letter of each word in the string into the uppercase and the other letters into the lowercase.

Return type: text

Example:

```
SELECT initcap('hi THOMAS');  
initcap  
-----  
Hi Thomas  
(1 row)
```

quote_ident(string text)

Description: Returns the given string suitably quoted to be used as an identifier in an SQL statement string (quotation marks are used as required). Quotation marks are added only when necessary (the string contains non-identifier characters or characters that can be converted to uppercase or lowercase). Embedded quotation marks are doubled.

Return type: text

Example:

```
SELECT quote_ident('hello world');
quote_ident
-----
"hello world"
(1 row)
```

quote_literal(string text)

Description: Returns the given string suitably quoted to be used as a text in an SQL statement string (quotation marks are used as required).

Return type: text

Example:

```
SELECT quote_literal('hello');
quote_literal
-----
'hello'
(1 row)
```

If command similar to the following exists, text will be escaped.

```
SELECT quote_literal(E'O\hello');
quote_literal
-----
'O"hello'
(1 row)
```

If command similar to the following exists, backslash will be properly doubled.

```
SELECT quote_literal('O\hello');
quote_literal
-----
E'O\\hello'
(1 row)
```

If the parameter is NULL, NULL is returned. If the parameter may be NULL, you are advised to use **quote_nullable**.

```
SELECT quote_literal(NULL);
quote_literal
-----
(1 row)
```

quote_literal(value anyelement)

Description: Coerces the given value to text and then quotes it as a literal.

Return type: text

Example:

```
SELECT quote_literal(42.5);
quote_literal
-----
'42.5'
(1 row)
```

If command similar to the following exists, the given value will be escaped.

```
SELECT quote_literal(E'O\42.5');
quote_literal
-----
'O"42.5'
(1 row)
```

If command similar to the following exists, backslash will be properly doubled.

```
SELECT quote_literal('O\42.5');
quote_literal
-----
E'O\\42.5'
(1 row)
```

quote_nullable(string text)

Description: Returns the given string suitably quoted to be used as a string literal in an SQL statement string (quotation marks are used as required).

Return type: text

Example:

```
SELECT quote_nullable('hello');
quote_nullable
-----
'hello'
(1 row)
```

If command similar to the following exists, text will be escaped.

```
SELECT quote_nullable(E'O\hello');
quote_nullable
-----
'O"hello'
(1 row)
```

If command similar to the following exists, backslash will be properly doubled.

```
SELECT quote_nullable('O\hello');
quote_nullable
-----
E'O\\hello'
(1 row)
```

If the parameter is **NULL**, **NULL** is returned.

```
SELECT quote_nullable(NULL);
quote_nullable
-----
NULL
(1 row)
```

quote_nullable(value anyelement)

Description: Converts the given value to text and then quotes it as a literal.

Return type: text

Example:

```
SELECT quote_nullable(42.5);
quote_nullable
-----
'42.5'
(1 row)
```

If command similar to the following exists, the given value will be escaped.

```
SELECT quote_nullable(E'O\42.5');
quote_nullable
-----
'O"42.5'
(1 row)
```

If command similar to the following exists, backslash will be properly doubled.

```
SELECT quote_nullable('O\42.5');
quote_nullable
-----
'E'O\\42.5'
(1 row)
```

If the parameter is NULL, NULL is returned.

```
SELECT quote_nullable(NULL);
quote_nullable
-----
NULL
(1 row)
```

BIN(n bigint)

Description: Converts the bigint type from decimal to binary and returns the result as a string. If the parameter contains NULL, NULL will be returned.

Return type: text

Example:

```
SELECT BIN(16) as result;
result
-----
10000
(1 row)
```

HEX(n)

Description: Returns the hexadecimal string of **n**. **n** can be an integer or string. If the parameter contains NULL, NULL will be returned.

Return type: text

Example:

```
SELECT HEX(255) as result;
result
-----
FF
(1 row)
SELECT HEX('abc') as result;
result
-----
616263
(1 row)
```

UNHEX(n)

Description: Performs the reverse operation of **HEX(n)**. *n* can be of the int type or a string. Each pair of hexadecimal digits in a parameter is considered as a number and converted into the character represented by the number. If the parameter contains NULL, NULL will be returned. This parameter is supported only by clusters of version 8.2.0 or later.

Return type: bytea

Example:

```
SELECT UNHEX('abc') as result;
result
-----
\x0abc
(1 row)
```

reverse(str)

Description: Returns a string in reverse order.

Return type: text

Example:

```
SELECT reverse('abcde');
reverse
-----
edcba
(1 row)
```

to_hex(number int or bigint)

Description: Converts a number to a hexadecimal number.

Return type: text

Example:

```
SELECT to_hex(2147483647);
to_hex
-----
7fffffff
(1 row)
```

translate(string text, from text, to text)

Description: Replaces any character in the original string that matches a character in the **from** set by the corresponding character in the **to** set. If **from** is longer than **to**, extra characters occurred in **from** are removed.

Parameters:

- **string**: original string to be processed.
- **from**: specifies the characters to be replaced.
- **to**: specifies the characters after replacement.

Return type: text

Example:

Set the **from** parameter to **143** and the **to** parameter **ax**. Replace **1** and **4** in the string **12345** with **a** and **x** respectively. Delete **3** because there is no corresponding replacement character. **a2x5** is returned.

```
SELECT translate('12345', '143', 'ax');
translate
-----
a2x5
(1 row)
```

ascii(string)

Description: Returns the ASCII code of the first character in a string.

Return type: integer

Example:

Return the ASCII code of the first character **x** in the string **xyz**.

```
SELECT ascii('xyz');
ascii
-----
120
(1 row)
```

convert(string bytea, src_encoding name, dest_encoding name)

Description: Converts the string **bytea** to **dest_encoding**. **src_encoding** specifies the source code encoding. **string** must be valid in this encoding.

Return type: bytea

Example:

```
SELECT convert('text_in_utf8', 'UTF8', 'GBK');
convert
-----
\x746578745f696e5f75746638
(1 row)
```

 NOTE

If the rule for converting between source to target encoding (for example, GBK and LATIN1) does not exist, the string is returned without conversion. See the **pg_conversion** system catalog for details.

Example:

```
show server_encoding;
server_encoding
-----
LATIN1
(1 row)

SELECT convert_from('some text', 'GBK');
convert_from
-----
some text
(1 row)

db_latin1=# SELECT convert_to('some text', 'GBK');
convert_to
-----
\x736f6d652074657874
(1 row)

db_latin1=# SELECT convert('some text', 'GBK', 'LATIN1');
convert
-----
\x736f6d652074657874
(1 row)
```

convert_from(string bytea, src_encoding name)

Description: Converts the string **bytea** using the coding mode of the database.

src_encoding specifies the source code encoding. **string** must be valid in this encoding.

Return type: text

Example:

```
SELECT convert_from('text_in_utf8', 'UTF8');
convert_from
-----
text_in_utf8
(1 row)
SELECT convert_from('\x6461746162617365','gbk');
convert_from
-----
database
(1 row)
```

convert_to(string text, dest_encoding name)

Description: Converts the given string into one whose encoding format is **dest_encoding**.

Return type: bytea

Example:

```
SELECT convert_to('some text', 'UTF8');
convert_to
-----
```

```
\x736f6d652074657874
(1 row)
SELECT convert_to('database', 'gbk');
       convert_to
-----
\x6461746162617365
(1 row)
```

chr(integer)

Description: Converts an integer to the corresponding ASCII character.

Return type: varchar

Example:

```
SELECT chr(65);
 chr
-----
 A
(1 row)
```

format(formatstr text [, str"any" [, ...]])

Description: Formats a string.

Return type: text

Example:

```
SELECT format('Hello %s, %1$s', 'World');
       format
-----
Hello World, World
(1 row)
```

md5(string)

Description: Encrypts a string using MD5 and returns a hexadecimal number.

NOTE

MD5 is not recommended for security purposes.

Return type: text

Example:

```
SELECT md5('data');
       md5
-----
8d777f385d3dfec8815d20f7496026dc
(1 row)
```

decode(string text, format text)

Description: Decodes binary data from textual representation.

Return type: bytea

Example:

```
SELECT decode('ZGF0YWJhc2U=', 'base64');
       decode
-----
```

```
-----
\x6461746162617365
(1 row)

SELECT convert_from('\x6461746162617365','utf-8');
convert_from
-----
database
(1 row)
```

encode(data bytea, format text)

Description: Encodes binary data into a textual representation.

Return type: text

Example:

```
SELECT encode('database', 'base64');
encode
-----
ZGF0YWJhc2U=
(1 row)
```

CONV(n, fromBase, toBase)

Description: Converts the given value or string into value of a specific number system and outputs the result as a string. If the parameter contains NULL, NULL will be returned. The value range is [-36, -2] and [2, 36].

Return type: text

Example:

```
SELECT CONV(-1, 10, 16) as result;
result
-----
FFFFFFFFFFFFFFFF
(1 row)
```

21.7.1.9 Comparing Strings

STRCMP(text, text)

Description: Compares two strings. If all strings are the same, **0** is returned. If the first string is less than the second string according to the current classification order, **-1** is returned. Otherwise, **1** is returned. If the parameter contains NULL, NULL will be returned.

Return type: text

Example:

```
SELECT STRCMP('AA', 'AA'), STRCMP('AA', 'AB'), STRCMP('AA', 'A');
STRCMP | STRCMP | STRCMP
-----
0 | -1 | 1
(1 row)
```

21.7.1.10 Obtaining the Position of a Character or Substring

instr(text,text,int,int)

Description: Obtains the position of a substring in the string. If no match is found, **0** is returned.

- First **text**: (mandatory) target string.
- Second **text**: (mandatory) substring to be matched.
- First **int**: start position of the matching. If this parameter is not set, the matching starts from the beginning by default.
- Second **int**: which occurrence of the substring the function should return the position for. If this parameter is not set, the first occurrence is used by default.

Return type: integer

Example:

Obtain the position of the 2nd occurrence of **bcd** starting from position 2 in the main string **abcdabcdabcd**.

```
SELECT instr( 'abcdabcdabcd', 'bcd', 2, 2 );
instr
-----
      6
(1 row)

SELECT instr( 'abcdabcdabcd', 'lmk', 2);
instr
-----
      0
(1 row)
```

position(substring in string)

Description: Returns the position of a specified substring in the string. If the string does not contain substrings, **0** is returned.

Return type: integer

Example:

```
SELECT position('ing' in 'string');
position
-----
      4
(1 row)

SELECT position('ing' in 'strin');
position
-----
      0
(1 row)
```

strpos(string, substring)

Description: Obtains the position of the first occurrence of the specified substring in the string. If the substring does not exist in the string, **0** is returned.

Return type: integer

Example:

```
SELECT strpos('source', 'rc');
-----
      4
(1 row)

SELECT strpos('source', 'data');
-----
      0
(1 row)
```

instr(string,substring[,position,occurrence])

Description: Queries and returns the value of the substring **position** that occurs the **occurrence** (first by default) times from the **position** (1 by default) of the **string**.

- If the value of **position** is **0**, **0** is returned.
- If the value of **position** is negative, searches backwards from the last *n*th character in the string, in which *n* indicates the absolute value of **position**.

The unit is character. One Chinese character is one character.

Return type: integer

Example:

```
SELECT instr('corporate floor','or', 3);
-----
      5
(1 row)
SELECT instr('corporate floor','or',-3,2);
-----
      2
(1 row)
```

locate(substring,string[,position])

Description: Queries and returns the value of the substring **position** that appears for the first time from the position specified by **position** (1 by default) of **string**. The unit is character. If no substring exists in the **string**, **0** is returned.

Return type: integer

Example:

```
SELECT locate('ball','football');
-----
      5
(1 row)
SELECT locate('er','soccerplayer','6');
-----
     11
(1 row)
```

21.7.1.11 Checking If a String Matches a Specific Pattern Using String Matching Functions

What Are Pattern Matching Functions?

LIKE is a string comparison function used to check whether a string matches a specified pattern rule. It returns a Boolean value (**true/false**, represented by **t/f** in the example).

The differences between **LIKE** and equality (=) operators are as follows:

- = matches the complete string exactly, including its length and content.
- **LIKE** performs a fuzzy match using wildcards to overlook minor character variations while focusing on the main pattern

Scenarios: These functions are mainly used for fuzzy query. For example:

- To find users whose names contain **Zhang**, use **name LIKE '%Zhang%'**;
- To find users whose mobile numbers start with **138**, use **phone LIKE '138%'**;
- To find users whose email addresses end with **@abc.com** and the user name length is **5**, use **email LIKE '____@abc.com'** (**five underscores**).

regexp_like(text,text,text)

Description: Performs a regular expression matching.

Return type: Boolean

Example:

```
SELECT regexp_like('str','[ac]');
regexp_like
-----
f
(1 row)
```

string [NOT] LIKE pattern [ESCAPE escape-character]

Description: Specifies the pattern matching function.

If the pattern contains neither percent signs nor underscores, it stands solely for itself, making **LIKE** behave like the equality operator. An underscore (_) in the pattern matches any single character, while a percent sign (%) matches zero or more characters.

To match an underscore or percent sign literally, the respective character in the pattern must be preceded by an escape character. The default escape character is a backward slash (\) and can be specified using the **ESCAPE** clause. To match with escape characters, enter two escape characters.

Return type: boolean

Example:

```
SELECT 'AA_BBCC' LIKE '%A@_B%' ESCAPE '@' AS RESULT;
result
-----
```

```
t
(1 row)
SELECT 'AA_BBCC' LIKE '%A@_B%' AS RESULT;
result
-----
f
(1 row)
SELECT 'AA@_BBCC' LIKE '%A@_B%' AS RESULT;
result
-----
t
(1 row)
```

REGEXP_LIKE(source_string, pattern [, match_parameter])

Description: Performs a regular expression matching.

source_string indicates the source string and **pattern** indicates the matching pattern of the regular expression. **match_parameter** indicates the matching items and the values are as follows:

- **i**: case-insensitive.
- **c**: case-sensitive
- **n**: allows the metacharacter "." in a regular expression to be matched with a newline chara
- **m**: allows **source_string** to be regarded as multiple rows.

If **match_parameter** is ignored, case-sensitive will be used by default, meaning that "." will not be matched with a newline character and **source_string** will be regarded as a single row.

Return type: boolean

Example:

```
SELECT regexp_like('ABC', '[A-Z]');
regexp_like
-----
t
(1 row)
SELECT regexp_like('ABC', '[D-Z]');
regexp_like
-----
f
(1 row)
SELECT regexp_like('abc', '[A-Z]', 'i');
regexp_like
-----
t
(1 row)
SELECT regexp_like('abc', '[A-Z]');
regexp_like
-----
f
(1 row)
```

21.7.2 Binary String Functions and Operators

SQL defines some binary string functions that use keywords, rather than commas, to separate arguments. DWS also provides the common syntax used for invoking functions.

octet_length(string)

Description: Returns the number of bytes in the given binary string.

Return type: integer

Example:

```
SELECT octet_length(E'jo\000se'::bytea) AS RESULT;  
result  
-----  
5  
(1 row)
```

overlay(string placing string from int [for int])

Description: Replaces the given substring.

Return type: bytea

Example:

```
SELECT overlay(E'Th\000omas'::bytea placing E'\002\003'::bytea from 2 for 3) AS RESULT;  
result  
-----  
\x5402036d6173  
(1 row)
```

position(substring in string)

Description: Returns the location of the given substring.

Return type: integer

Example:

```
SELECT position(E'\000om'::bytea in E'Th\000omas'::bytea) AS RESULT;  
result  
-----  
3  
(1 row)
```

substring(string [from int] [for int])

Description: Truncates the given substring.

Return type: bytea

Example:

```
SELECT substring(E'Th\000omas'::bytea from 2 for 3) AS RESULT;  
result  
-----  
\x68006f  
(1 row)
```

Truncate the time and obtain the number of hours.

```
SELECT substring('2022-07-18 24:38:15',12,2)AS RESULT;  
result  
-----  
24  
(1 row)
```

trim([both] bytes from string)

Description: Removes the longest string containing only bytes from **bytes** from the start and end of **string**.

Return type: bytea

Example:

```
SELECT trim(E'\000':bytea from E'\000Tom\000':bytea) AS RESULT;  
result  
-----  
\x546f6d  
(1 row)
```

substring_index(string, delim, count)

Description: Searches for delimiters in a case-sensitive manner and returns the substring before the delimiter that appears for the **count** time. If **count** is a negative number, count for the delimiter reversely from the end. If the parameter contains NULL, NULL will be returned. This function is supported by version 8.2.0 or later clusters.

Return type: text

Example: In the string **www.wWw.cloud.wWw.com**, find the delimiter **.wWw.** that appears the second time in a case-sensitive manner. Return the substring before it: **www.wWw.cloud**.

```
SELECT SUBSTRING_INDEX('www.wWw.cloud.wWw.com', '.wWw.', 2) AS RESULT;  
result  
-----  
www.wWw.cloud  
(1 row)
```

btrim(string bytea, bytes bytea)

Description: Removes the longest string containing only bytes from **bytes** from the start and end of **string**.

Return type: bytea

Example:

```
SELECT btrim(E'\000trim\000':bytea, E'\000':bytea) AS RESULT;  
result  
-----  
\x7472696d  
(1 row)
```

get_bit(string, offset)

Description: Returns the number of bits in the given string.

Return type: integer

Example:

```
SELECT get_bit(E'Th\000omas':bytea, 45) AS RESULT;  
result  
-----  
1  
(1 row)
```

get_byte(string, offset)

Description: Returns the number of bytes in the given string.

Return type: integer

Example:

```
SELECT get_byte(E'Th\\000omas':bytea, 4) AS RESULT;  
result  
-----  
109  
(1 row)
```

set_bit(string,offset, newvalue)

Description: Sets bits in the given string.

Return type: bytea

Example:

```
SELECT set_bit(E'Th\\000omas':bytea, 45, 0) AS RESULT;  
result  
-----  
\x5468006f6d4173  
(1 row)
```

set_byte(string,offset, newvalue)

Description: Sets bytes in the given string.

Return type: bytea

Example:

```
SELECT set_byte(E'Th\\000omas':bytea, 4, 64) AS RESULT;  
result  
-----  
\x5468006f406173  
(1 row)
```

21.7.3 Bit String Functions and Operators

Aside from the usual comparison operators, the following operators can be used. Bit string operands of **&**, **|**, and **#** must be of equal length. When bit shifting, the original length of the string is preserved by zero padding (if necessary).

||

Description: Connects bit strings.

For example:

```
SELECT B'10001' || B'011' AS RESULT;  
result  
-----  
10001011  
(1 row)
```

&

Description: AND operation between bit strings

For example:

```
SELECT B'10001' & B'01101' AS RESULT;  
result  
-----  
00001  
(1 row)
```

|

Description: OR operation between bit strings

For example:

```
SELECT B'10001' | B'01101' AS RESULT;  
result  
-----  
11101  
(1 row)
```

#

Description: OR operation between bit strings if they are inconsistent. If the same positions in the two bit strings are both 1 or 0, the position returns **0**.

For example:

```
SELECT B'10001' # B'01101' AS RESULT;  
result  
-----  
11100  
(1 row)
```

~

Description: NOT operation between bit strings

For example:

```
SELECT ~B'10001' AS RESULT;  
result  
-----  
01110  
(1 row)
```

<<

Description: binary left shift

For example:

```
SELECT B'10001' << 3 AS RESULT;  
result  
-----  
01000  
(1 row)
```

>>

Description: binary right shift

For example:

```
SELECT B'10001' >> 2 AS RESULT;  
result
```

```
-----  
00100  
(1 row)
```

The following SQL-standard functions work on bit strings as well as character strings: **length**, **bit_length**, **octet_length**, **position**, **substring**, and **overlay**.

The following functions work on bit strings as well as binary strings: **get_bit** and **set_bit**. When working with a bit string, these functions number the first (leftmost) bit of the string as bit 0.

Integers and bits can be mutually converted. For example:

```
SELECT 44::bit(10) AS RESULT;  
result  
-----  
0000101100  
(1 row)  
  
SELECT 44::bit(3) AS RESULT;  
result  
-----  
100  
(1 row)  
  
SELECT cast(-44 as bit(12)) AS RESULT;  
result  
-----  
11111010100  
(1 row)  
  
SELECT '1110'::bit(4)::integer AS RESULT;  
result  
-----  
14  
(1 row)
```

NOTE

Casting to just "bit" means casting to bit(1), and so will deliver only the least significant bit of the integer.

21.7.4 Numeric Functions and Operators

21.7.4.1 Numeric Operators

+

Description: addition

Example:

```
SELECT 2+3 AS RESULT;  
result  
-----  
5  
(1 row)
```

-

Description: subtraction

Example:

```
SELECT 2-3 AS RESULT;  
result  
-----  
    -1  
(1 row)
```

*

Description: multiplication

Example:

```
SELECT 2*3 AS RESULT;  
result  
-----  
     6  
(1 row)
```

/

Description: division (The result is not rounded.)

Example:

```
SELECT 4/2 AS RESULT;  
result  
-----  
     2  
(1 row)  
SELECT 4/3 AS RESULT;  
result  
-----  
1.3333333333333333  
(1 row)
```

+/-

Description: positive/negative

Example:

```
SELECT -2 AS RESULT;  
result  
-----  
    -2  
(1 row)
```

%

Description: modulo (to obtain the remainder)

Example:

```
SELECT 5%4 AS RESULT;  
result  
-----  
     1  
(1 row)
```

@

Description: absolute value

Example:

```
SELECT @ -5.0 AS RESULT;  
result  
-----  
5.0  
(1 row)
```

^

Description: power (exponent calculation)

In MySQL-compatible mode, this operator means exclusive OR. For details, see operator **#** in [Bit String Functions and Operators](#).

Example:

```
SELECT 2.0^3.0 AS RESULT;  
result  
-----  
8.000000000000000  
(1 row)
```

||

Description: square root

Example:

```
SELECT || 25.0 AS RESULT;  
result  
-----  
5  
(1 row)
```

|||

Description: cubic root

Example:

```
SELECT ||| 27.0 AS RESULT;  
result  
-----  
3  
(1 row)
```

!

Description: factorial

Example:

```
SELECT 5! AS RESULT;  
result  
-----  
120  
(1 row)
```

!!

Description: factorial (prefix operator)

Example:

```
SELECT !!5 AS RESULT;  
result  
-----  
120  
(1 row)
```

&

Description: binary AND

Example:

```
SELECT 91&15 AS RESULT;  
result  
-----  
11  
(1 row)
```

|

Description: binary OR

Example:

```
SELECT 32|3 AS RESULT;  
result  
-----  
35  
(1 row)
```

#

Description: binary XOR

Example:

```
SELECT 17#5 AS RESULT;  
result  
-----  
20  
(1 row)
```

~

Description: binary NOT

Example:

```
SELECT ~1 AS RESULT;  
result  
-----  
-2  
(1 row)
```

<<

Description: binary shift left

Example:

```
SELECT 1<<4 AS RESULT;  
result  
-----
```



```
16  
(1 row)
```

>>

Description: binary shift right

Example:

```
SELECT 8>>2 AS RESULT;  
result  
-----  
2  
(1 row)
```

21.7.4.2 Numeric Functions

abs(x)

Description: Returns the absolute value.

Return type: same as the input

Example:

```
SELECT abs(-17.4);  
abs  
-----  
17.4  
(1 row)
```

acos(x)

Description: Returns the arccosine.

Return type: double precision

Example:

```
SELECT acos(-1);  
acos  
-----  
3.14159265358979  
(1 row)
```

asin(x)

Description: Returns the arcsine.

Return type: double precision

Example:

```
SELECT asin(0.5);  
asin  
-----  
0.523598775598299  
(1 row)
```

atan(x)

Description: Returns the arctangent.

Return type: double precision

Example:

```
SELECT atan(1);
      atan
-----
0.785398163397448
(1 row)
```

atan2(y, x)

Description: Returns the arctangent of y/x.

Return type: double precision

Example:

```
SELECT atan2(2, 1);
      atan2
-----
1.10714871779409
(1 row)
```

bitand(integer, integer)

Description: Performs AND (&) operation on two integers.

Return type: bigint

Example:

```
SELECT bitand(127, 63);
      bitand
-----
63
(1 row)
```

cbrt(double precision)

Description: Returns the cubic root.

Return type: double precision

Example:

```
SELECT cbrt(27.0);
      cbrt
-----
3
(1 row)
```

ceil(double precision or numeric)

Description: Returns the minimum integer greater than or equal to a specified number.

Return type: same as the input

Example:

```
SELECT ceil(-42.8);
      ceil
-----
```

```
-42  
(1 row)
```

ceiling(double precision or numeric)

Description: Returns the minimum integer greater than or equal to a specified number. This function is the alias of the ceil(x) function.

Return type: same as the input

Example:

```
SELECT ceiling(-95.3);  
ceiling  
-----  
-95  
(1 row)
```

cos(x)

Description: Returns the cosine.

Return type: double precision

Example:

```
SELECT cos(-3.1415927);  
cos  
-----  
-0.9999999999999999  
(1 row)
```

cot(x)

Description: Returns the cotangent.

Return type: double precision

Example:

```
SELECT cot(1);  
cot  
-----  
0.642092615934331  
(1 row)
```

degrees(double precision)

Description: Converts radians to angles.

Return type: double precision

Example:

```
SELECT degrees(0.5);  
degrees  
-----  
28.6478897565412  
(1 row)
```

div(y numeric, x numeric)

Description: Returns the integer part of y divided by x.

Return type: numeric

Example:

```
SELECT div(9,4);
div
-----
2
(1 row)
```

exp(double precision or numeric)

Description: Returns the natural exponent.

Return type: same as the input

Example:

```
SELECT exp(1.0);
exp
-----
2.7182818284590452
(1 row)
```

floor(double precision or numeric)

Description: Returns the maximum integer less than or equal to a specified number.

Return type: same as the input

Example:

```
SELECT floor(-42.8);
floor
-----
-43
(1 row)
```

radians(double precision)

Description: Converts angles to radians.

Return type: double precision

Example:

```
SELECT radians(45.0);
radians
-----
0.785398163397448
(1 row)
```

random()

Description: Returns a random number between 0.0 and 1.0.

Return type: double precision

Example:

```
SELECT random();
random
-----
```

```
0.824823560658842  
(1 row)
```

rand()

Description: Returns a random number between 0.0 and 1.0. This function is compatible with MySQL. This function is supported by version 8.2.0 or later clusters.

Return type: double precision

Example:

```
SELECT rand();  
      rand  
-----  
0.824823560658842  
(1 row)
```

ln(double precision or numeric)

Description: Returns the natural logarithm.

Return type: same as the input

Example:

```
SELECT ln(2.0);  
      ln  
-----  
0.6931471805599453  
(1 row)
```

log(double precision or numeric)

Description: Returns the logarithm with base 10.

- In the ORA- or TD-compatible mode, this function returns the logarithm with base 10.
- In the MySQL-compatible mode, this function returns the natural logarithm.

Return type: same as the input

Example:

```
-- ORA-compatible mode  
SELECT log(100.0);  
      log  
-----  
2.0000000000000000  
(1 row)  
-- TD-compatible mode  
SELECT log(100.0);  
      log  
-----  
2.0000000000000000  
(1 row)  
-- MySQL-compatible mode  
SELECT log(100.0);  
      log  
-----  
4.6051701859880914  
(1 row)
```

log(b numeric, x numeric)

Description: Returns the logarithm with base b.

Return type: numeric

Example:

```
SELECT log(2.0, 64.0);
      log
-----
6.0000000000000000
(1 row)
```

mod(x,y)

Description: Specifies the remainder of x/y (modulus). If x is 0, 0 is returned. If y is 0, x is returned.

Return type: same as the parameter type

Example:

```
SELECT mod(9,4);
      mod
-----
1
(1 row)
SELECT mod(9,0);
      mod
-----
9
(1 row)
```

pi()

Description: Returns constant π .

Return type: double precision

Example:

```
SELECT pi();
      pi
-----
3.14159265358979
(1 row)
```

power(a double precision, b double precision)

Description: Returns a raised to the power of b.

Return type: double precision

Example:

```
SELECT power(9.0, 3.0);
      power
-----
729.0000000000000000
(1 row)
```

round(double precision or numeric)

Description: Returns the integer closest to the input parameter.

Return type: same as the input

Example:

```
SELECT round(42.4);
round
-----
    42
(1 row)

SELECT round(42.6);
round
-----
    43
(1 row)
```

NOTE

When the **round** function is invoked, the numeric type is rounded to zero. While on most computers, the real number and the double-precision number are rounded to the nearest even number.

round(v numeric, s int)

Description: Rounds to **s** decimal places.

Return type: numeric

Example:

```
SELECT round(42.4382, 2);
round
-----
 42.44
(1 row)
```

setseed(double precision)

Description: Sets seed for the following random() invoking (between -1.0 and 1.0, inclusive).

Return type: void

Example:

```
SELECT setseed(0.54823);
setseed
-----
(1 row)
```

sign(double precision or numeric)

Description: Returns symbols of this parameter.

Return type: **-1** indicates a negative number, **0** indicates 0, and **1** indicates a positive number.

Example:

```
SELECT sign(-8.4);
sign
-----
-1
(1 row)
```

sin(x)

Description: Returns the sine.

Return type: double precision

Example:

```
SELECT sin(1.57079);
sin
-----
0.999999999979986
(1 row)
```

sqrt(double precision or numeric)

Description: Returns the square root.

Return type: same as the input

Example:

```
SELECT sqrt(2.0);
sqrt
-----
1.414213562373095
(1 row)
```

tan(x)

Description: Returns the tangent.

Return type: double precision

Example:

```
SELECT tan(20);
tan
-----
2.23716094422474
(1 row)
```

trunc(double precision or numeric)

Description: Returns the integer part of a number.

Return type: same as the input

Example:

```
SELECT trunc(42.8);
trunc
-----
42
(1 row)
```


trunc(v numeric, s int)

Description: Truncates a number to **s** decimal places.

Return type: numeric

Example:

```
SELECT trunc(42.4382, 2);
trunc
-----
42.43
(1 row)
```

truncate(v numeric, s int)

Description: Truncates **v** to **s** decimal places. In addition to any precision, **v** can also be an integer or float number. The return value type is the same as that of the input parameter **v**. If **s** is a negative number, the integer part is truncated. This function is supported by version 8.2.0 or later clusters.

Return type: numeric

Example:

```
SELECT truncate(42.4382, 2);
truncate
-----
42.4300
(1 row)
```

width_bucket(operand numeric, b1 numeric, b2 numeric, count int)

Description: Sets the minimum value, maximum value, and number of groups in a group range, constructs a specified number of groups with the same size, and returns the ID of the group to which a specified field value belongs. **b1** is the minimum value of the group range, **b2** is the maximum value of the group range, and **count** is the number of groups.

Return type: integer

Example:

```
SELECT width_bucket(5.35, 0.024, 10.06, 5);
width_bucket
-----
3
(1 row)
```

width_bucket(operand double precision, b1 double precision, b2 double precision, count int)

Description: Sets the minimum value, maximum value, and number of groups in a group range, constructs a specified number of groups with the same size, and returns the ID of the group to which a specified field value belongs. **b1** is the minimum value of the group range, **b2** is the maximum value of the group range, and **count** is the number of groups.

Return type: integer

Example:

```
SELECT width_bucket(5.35, 0.024, 10.06, 5);
width_bucket
-----
3
(1 row)
```

21.7.5 Date and Time Processing Functions and Operators

21.7.5.1 Date and Time Operators

NOTICE

When you use date and time operators, add prefixes of explicit types to the operands to ensure that the operands can be properly parsed by the database.

For example, the operand in the following example does not have an explicit data type, and an error will occur.

```
SELECT date '2001-10-01' - '7' AS RESULT;
```

Table 21-32 Date and time operators

| Operator | Example | Result: |
|----------|--|---|
| + | date + integer → timestamp
Add days to a date.
SELECT date '2001-09-28' + integer '7' AS RESULT; | Add a date with an integer to obtain the time after 7 days.
2001-10-05 00:00:00 |
| + | date + interval → timestamp
Add an interval to a date.
SELECT date '2001-09-28' + interval '1 hour' AS RESULT;
SELECT date '2021-02-28' + interval '1 month' AS RESULT;
SELECT date '2021-01-31' + interval '1 month' AS RESULT; | Sum of the date parameter and the interval parameter
Obtain the time after one hour.
2001-09-28 01:00:00
Obtain the time after one month.
2021-02-28 00:00:00
If the sum or subtraction results fall beyond the date range of a month, the result will be rounded to the last day of the month.
2021-03-28 00:00:00 |
| + | date + time → timestamp
Add the time of a day to a date.
SELECT date '2001-09-28' + time '03:00' AS RESULT; | Add a date and a time to obtain a specific time.
2001-09-28 03:00:00 |
| + | interval + interval → interval
Add an interval.
SELECT interval '1 day' + interval '1 hour' AS RESULT; | Add two intervals to obtain the sum.
1 day 01:00:00 |

| Operator | Example | Result: |
|----------|--|--|
| + | timestamp + interval → timestamp
Add an interval to a timestamp.
SELECT timestamp '2001-09-28 01:00' + interval '23 hours' AS RESULT; | Add a timestamp and an interval to obtain the time after 23 hours.
2001-09-29 00:00:00 |
| + | time + interval → time
Add an interval to a time.
SELECT time '01:00' + interval '3 hours' AS RESULT; | Add a time and an interval to obtain the time after three hours.
04:00:00 |
| - | date - date → integer
Subtract a date from another to obtain the time difference.
SELECT date '2001-10-01' - date '2001-09-28' AS RESULT; | Subtract a date from another to obtain the time difference.
3 days |
| - | date - integer → timestamp
Subtract days from a date.
SELECT date '2001-10-01' - integer '7' AS RESULT; | Subtract an integer from a date to obtain the time difference between them.
2001-09-24 00:00:00 |
| - | date - interval → timestamp
Subtract an interval from a date.
SELECT date '2001-09-28' - interval '1 hour' AS RESULT; | Subtract an interval from a date to obtain the date and time difference.
2001-09-27 23:00:00 |
| - | time - time → interval
Subtract a time.
SELECT time '05:00' - time '03:00' AS RESULT; | Subtract a time from another time to obtain the time difference.
02:00:00 |
| - | time - interval → time
Subtract an interval from a time.
SELECT time '05:00' - interval '2 hours' AS RESULT; | Subtract an interval from a time to obtain the time difference.
03:00:00 |
| - | timestamp - interval → timestamp
Subtract an interval from a timestamp.
SELECT timestamp '2001-09-28 23:00' - interval '23 hours' AS RESULT; | Subtract an interval from a timestamp to obtain the date and time difference.
2001-09-28 00:00:00 |
| - | interval - interval → interval
Subtract an interval.
SELECT interval '1 day' - interval '1 hour' AS RESULT; | Subtract an interval from another interval to obtain the time difference.
23:00:00 |
| - | timestamp - timestamp → interval
Subtract a timestamp.
SELECT timestamp '2001-09-29 03:00' - timestamp '2001-09-27 12:00' AS RESULT; | Subtract a timestamp from another timestamp to obtain the time difference.
1 day 15:00:00 |

| Operator | Example | Result: |
|----------|--|---|
| - | <p>select now() - interval → timestamp with time zone</p> <p>Obtain the time on the previous day.</p> <pre>select now() - interval '1 day' AS RESULT;</pre> | <p>Obtain the time on the previous day.</p> <pre>2025-07-29 10:54:29.116501+08</pre> |
| * | <p>interval * double precision → interval</p> <p>Multiply an interval by a quantity.</p> <pre>SELECT 900 * interval '1 second' AS RESULT;
SELECT 21 * interval '1 day' AS RESULT;
SELECT double precision '3.5' * interval '1 hour' AS RESULT;</pre> | <p>Multiply the interval of 1 second by 900.</p> <pre>00:15:00</pre> <p>Multiply the interval of one day by 21.</p> <pre>21 days</pre> <p>Multiply the interval of one hour by 3.5.</p> <pre>03:30:00</pre> |
| / | <p>interval / double precision → interval</p> <pre>SELECT interval '1 hour' / double precision '1.5' AS RESULT;</pre> | <p>Divide an interval by a quantity to obtain a time segment.</p> <pre>00:40:00</pre> |

21.7.5.2 Date and Time Functions

age(timestamp, timestamp)

Description: Subtracts arguments and returns the difference. If the result is negative, the returned result is also negative.

Return type: interval

Example:

```
SELECT age(timestamp '2001-04-10', timestamp '1957-06-13');  
age  
-----  
43 years 9 mons 27 days  
(1 row)
```

age(timestamp)

Description: Subtracts the argument from the current date.

Return type: interval

Example:

```
SELECT age(timestamp '1957-06-13');  
age  
-----  
60 years 2 mons 18 days  
(1 row)
```

adddate(date, interval | int)

Description: Returns the result of a given datetime plus the time interval of a specified unit. The default unit is day (when the second parameter is an integer).

Return type: timestamp

Example:

When the input parameter is of the text type:

```
select adddate('2020-11-13', 10);
      adddate
-----
2020-11-23
(1 row)

select adddate('2020-11-13', interval '1' month);
      adddate
-----
2020-12-13
(1 row)

select adddate('2020-11-13 12:15:16', interval '1' month);
      adddate
-----
2020-12-13 12:15:16
(1 row)

select adddate('2020-11-13', interval '1' minute);
      adddate
-----
2020-11-13 00:01:00
(1 row)
```

When the input parameter is of the date type:

```
select adddate(current_date, 10);
      adddate
-----
2021-09-24
(1 row)

select adddate(date '2020-11-13', interval '1' month);
      adddate
-----
2020-12-13 00:00:00
(1 row)
```

addtime(timestamp | time | text, interval | text)

Description: Returns the result of a given datetime plus the time interval of a specified unit. This function is supported by version 8.2.0 or later clusters.

Return type: same as the type of the first input parameter.

Example:

```
SELECT ADDTIME('2020-11-13 01:01:01', '23:59:59');
      addtime
-----
2020-11-14 01:01:00
(1 row)
```

addtime(time, interval)

```
SELECT ADDTIME('20:1:1'::TIME, '20:59:59+8');
      addtime
-----
233:01:00
(1 row)
```

```
SELECT ADDTIME('20:1:1'::TIME, '8 20:59:59');
      addtime
-----
233:01:00
(1 row)
```

addtime(text, interval)

```
SELECT ADDTIME('20:1:1', '20:59:59+8');
      addtime
-----
233:01:00
(1 row)
SELECT ADDTIME('20:1:1', '8 20:59:59');
      addtime
-----
233:01:00
(1 row)
```

subdate(date, interval | int)

Description: Returns the result of a given datetime minus the time interval of a specified unit. The default unit is day (when the second parameter is an integer).

Return type: timestamp

Example:

When the input parameter is of the text type:

```
select subdate('2020-11-13', 10);
      subdate
-----
2020-11-03
(1 row)

select subdate('2020-11-13', interval '2' month);
      subdate
-----
2020-09-13
(1 row)

select subdate('2020-11-13 12:15:16', interval '1' month);
      subdate
-----
2020-10-13 12:15:16
(1 row)

select subdate('2020-11-13', interval '2' minute);
      subdate
-----
2020-11-12 23:58:00
(1 row)
```

When the input parameter is of the date type:

```
select subdate(current_date, 10);
      subdate
-----
2021-09-05
(1 row)
```

```
select subdate(current_date, interval '1' month);
      subdate
-----
2021-08-15 00:00:00
(1 row)
```

subtime(timestamp | time | text, interval | text)

Description: Returns the result of a given datetime minus the time interval of a specified unit. This function is supported by version 8.2.0 or later clusters.

Return type: same as the type of the first input parameter.

Example:

```
select subtime('2020-11-13 01:01:01', '23:59:59');
      addtime
-----
2020-11-12 01:01:02
(1 row)
```

date_add(date, interval)

Description: Returns the result of a given datetime plus the time interval of a specified unit. It is equivalent to [adddate\(date, interval | int\)](#).

Return type: timestamp

date_sub(date, interval)

Description: Returns the result of a given datetime minus the time interval of a specified unit. It is equivalent to [subdate\(date, interval | int\)](#).

Return type: timestamp

timestampadd(field, numeric, timestamp)

Description: Adds an integer interval in the unit of **field** (the number of seconds can be a decimal) to a datetime expression. If the value is negative, the corresponding time interval is subtracted from the given datetime expression. The **field** can be **year**, **month**, **quarter**, **day**, **week**, **hour**, **minute**, **second**, or **microsecond**.

NOTE

When you pass a time expression in the **yyyy-mon-day+{value}** format to the function, the **{value}** part represents the time zone for the given date. For example, **2023-01-31+1** means January 31, 2023, in the UTC+1 time zone. In DWS, this format is handled differently in MySQL. MySQL processes this format only if the loose parsing rule is enabled, interpreting it as 01:00 on January 31, 2023, in the system's default time zone.

Return type: timestamp

Example:

```
select timestampadd(year, 1, timestamp '2020-2-29');
      timestampadd
-----
2021-02-28 00:00:00
(1 row)
```

```
select timestampadd(second, 2.354156, timestamp '2020-11-13');
timestampadd
-----
2020-11-13 00:00:02.354156
(1 row)
```

timestampdiff(field, timestamp1, timestamp2)

Description: Subtracts **timestamp1** from **timestamp2** and returns the difference in the unit of **field**. If the difference is negative, this function returns it normally. The **field** can be **year**, **month**, **quarter**, **day**, **week**, **hour**, **minute**, **second**, or **microsecond**.

Return type: bigint

Example:

```
SELECT timestampdiff(day, timestamp '2001-02-01', timestamp '2003-05-01 12:05:55');
timestampdiff
-----
819
(1 row)
```

timediff(timestamp | time | text, timestamp | time | text)

Description: Subtracts a date from another date. If the difference is negative, this function returns it normally. The types of the two input parameters must be the same. This function is supported by version 8.2.0 or later clusters.

Return type: time

Example:

```
SELECT timediff('2022-7-5 1:1:1', '2021-7-5 1:1:1');
timediff
-----
8760:00:00
(1 row)
```

clock_timestamp()

Description: Returns the current timestamp of the real-time clock.

Return type: timestamp with time zone

Example:

```
SELECT clock_timestamp();
clock_timestamp
-----
2017-09-01 16:57:36.636205+08
(1 row)
```

current_date

Description: Returns the current date.

Return type: date

Example:

```
SELECT current_date;
date
-----
```



```
-----  
2017-09-01  
(1 row)
```

curdate()

Description: Returns the current date. This function is compatible with MySQL. This function is supported by version 8.2.0 or later clusters.

Return type: date

Example:

```
SELECT curdate();  
curdate  
-----  
2022-09-19  
(1 row)
```

current_time

Description: Returns the current time.

Return type: time with time zone

Example:

```
SELECT current_time;  
timetz  
-----  
16:58:07.086215+08  
(1 row)
```

curtime([fsp])

Description: Returns the current time. **fsp** is an optional parameter. Its value is an integer, which indicates the precision of the returned result. This function is supported by version 8.2.0 or later clusters.

Return type: time with time zone

Example:

```
SELECT curtime();  
timetz  
-----  
16:58:07.086215+08  
(1 row)  
SELECT curtime(2);  
timetz  
-----  
16:58:07.08+08  
(1 row)
```

current_timestamp

Description: Returns the current date and time.

Return type: timestamp with time zone

Example:

```
SELECT current_timestamp;  
pg_systimestamp
```

```
-----  
2017-09-01 16:58:19.22173+08  
(1 row)
```

convert_tz(timestamp, from_tz, to_tz)

Description: Converts the datetime value from the time zone provided by **from_tz** (text) to the time zone provided by **to_tz** (text), and returns the converted datetime. This function is supported by version 8.2.0 or later clusters.

Return type: timestamp without time zone

Example:

```
SELECT convert_tz('2018-12-25 13:25:00', '+02:00', '+08:00');  
convert_tz  
-----  
2018-12-25 19:25:00  
(1 row)  
SELECT convert_tz('2000-02-28 23:00:00', 'GMT', 'MET');  
convert_tz  
-----  
2000-02-29 00:00:00  
(1 row)
```

datediff(date1, date2)

Description: Returns the number of days between two given dates.

Return type: bigint

Example:

```
select datediff(date '2020-11-13', date '2012-10-16');  
datediff  
-----  
2950  
(1 row)
```

date_part(text, timestamp)

Description: Obtains the precision specified by **text**.

This function is equivalent to **extract(field from timestamp)**.

Return type: double precision

Example:

```
SELECT date_part('hour', timestamp '2001-02-16 20:38:40');  
date_part  
-----  
20  
(1 row)
```

date_part(text, interval)

Description: Obtains the precision specified by **text**. If the value is greater than 12, the remainder of the value divided by 12 is returned.

This function is equivalent to **extract(field from timestamp)**.

Return type: double precision

Example:

```
SELECT date_part('month', interval '2 years 3 months');
date_part
-----
      3
(1 row)
```

date_trunc(text, timestamp with time zone | timestamp without time zone | interval)

Description: Truncates the time and date data to the specified precision (**text**). **text** supports microseconds, milliseconds, second, minute, hour, day, week, month, quarter, year, decade, century and millennium.

Return type: timestamp with time zone, timestamp without time zone, and interval

Example:

Truncate the input time to the hour.

```
SELECT date_trunc('hour', timestamp '2025-02-16 20:38:40');
date_trunc
-----
2025-02-16 20:00:00
(1 row)
```

Truncate the input time to the day.

```
SELECT date_trunc('day', timestamptz '2025-02-16 20:38:40+08');
date_trunc
-----
2025-02-16 00:00:00-05
(1 row)
```

Truncate the input time to the month.

```
SELECT date_trunc('month', timestamp '2025-02-16 20:38:40');
date_trunc
-----
2025-02-01 00:00:00
(1 row)
```

Truncate the input timestamp to the first day of a quarter.

```
SELECT DATE_TRUNC('quarter', TIMESTAMP '20250430 04:05:06.789');
date_trunc
-----
2025-04-01 00:00:00
(1 row)
```

Truncate the first day of the current month at 12:00.

```
SELECT date_trunc('month', now()) + interval '12h';
?column?
-----
2025-09-01 12:00:00-04
(1 row)
```

Truncate 09:00 of the current day.

```
SELECT date_trunc('day', now()) + interval '9h';
?column?
-----
2025-09-22 09:00:00-04
(1 row)
```

Truncate the first day of the current year.

```
SELECT date_trunc('year',CURRENT_DATE);
       date_trunc
-----
2025-01-01 00:00:00-05
(1 row)
```

trunc(timestamp)

Description: Truncates a timestamp to day.

Return type: timestamp

Example:

```
SELECT trunc(timestamp '2001-02-16
20:38:40');
       trunc
-----
2001-02-16 00:00:00
(1 row)
```

extract(field from timestamp)

Description: Obtains the value of **field** with the specified precision. For details about the valid values of **field**, see [EXTRACT](#).

Return type: double precision

Example:

```
SELECT extract(hour from timestamp '2001-02-16 20:38:40');
       date_part
-----
20
(1 row)
```

extract(field from interval)

Description: Obtains the value of **field** with the specified precision. If the value is greater than 12, the remainder of the value divided by 12 is returned. For details about the valid values of **field**, see [EXTRACT](#).

Return type: double precision

Example:

```
SELECT extract(month from interval '2 years 3 months');
       date_part
-----
3
(1 row)
```

day(date)

Description: Returns the number of days in the month in which **date** is located. This function is the same as the **dayofmonth** function.

Value range: 1 to 31

Return type: integer

Example:

```
select day('2020-06-28');
day
-----
28
(1 row)
```

dayofmonth(date)

Description: Returns the number of days in the month in which **date** is located.

Value range: 1 to 31

Return type: integer

Example:

```
select dayofmonth('2020-06-28');
dayofmonth
-----
28
(1 row)
```

dayofweek(date)

Description: Returns the week index corresponding to the given date, with Sunday as the start day of the week.

Value range: 1 to 7

Return type: integer

Example:

```
select dayofweek('2020-11-22');
dayofweek
-----
1
(1 row)
```

dayofyear(date)

Description: Returns the number of days of a given date in the current year.

Value range: 1 to 366

Return type: integer

Example:

```
select dayofyear('2020-02-29');
dayofyear
-----
60
(1 row)
```

hour(timestamp with time zone)

Description: Returns the hour value in a timestamp.

Return type: integer

Example:

```
SELECT hour(timestampz '2018-12-13 12:11:15+06');
hour
-----
14
(1 row)
```

isfinite(date)

Description: Checks whether a date is valid.

Return type: boolean

Example:

```
SELECT isfinite(date '2001-02-16');
isfinite
-----
t
(1 row)
SELECT isfinite(date 'infinity');
isfinite
-----
f
(1 row)
```

isfinite(timestamp)

Description: Checks whether a timestamp is valid.

Return type: boolean

Example:

```
SELECT isfinite(timestamp '2001-02-16 21:28:30');
isfinite
-----
t
(1 row)
SELECT isfinite(timestamp 'infinity');
isfinite
-----
f
(1 row)
```

isfinite(interval)

Description: Checks whether an interval is valid.

Return type: boolean

Example:

```
SELECT isfinite(interval '4 hours');
isfinite
-----
t
(1 row)
```

justify_days(interval)

Description: Adjusts an interval so that 30-day time periods are represented as months.

Return type: interval

Example:

```
SELECT justify_days(interval '35 days');
justify_days
-----
1 mon 5 days
(1 row)
```

justify_hours(interval)

Description: Adjusts an interval so that 24-hour time periods are represented as days.

Return type: interval

Example:

```
SELECT JUSTIFY_HOURS(INTERVAL '27 HOURS');
justify_hours
-----
1 day 03:00:00
(1 row)
```

justify_interval(interval)

Description: Adjusts an interval using **justify_days** and **justify_hours**.

Return type: interval

Example:

```
SELECT JUSTIFY_INTERVAL(INTERVAL '1 MON -1 HOUR');
justify_interval
-----
29 days 23:00:00
(1 row)
```

localtime

Description: Returns the current time.

Return type: time

Example:

```
SELECT localtime AS RESULT;
result
-----
16:05:55.664681
(1 row)
```

localtimestamp

Description: Returns the current date and time.

Return type: timestamp

Example:

```
SELECT localtimestamp;
timestamp
-----
2017-09-01 17:03:30.781902
(1 row)
```

makedate(year, dayofyear)

Description: Returns a date value based on the given year and the number of days in a year.

Return type: date

Example:

```
select makedate(2020, 60);
makedate
-----
2020-02-29
(1 row)
```

maketime(hour, minute, second)

Description: Returns a value of the time type based on the given hour, minute, and second. The value of the time type in DWS ranges from 00:00:00 to 24:00:00. Therefore, this function is not available when the value of hour is greater than 24 or less than 0.

Return type: time

Example:

```
select maketime(12, 15, 30.12);
maketime
-----
12:15:30.12
(1 row)
```

microsecond(timestamp with time zone)

Description: Returns the microsecond value in the time.

Return type: integer

Example:

```
SELECT microsecond(timestampz '2018-12-13 12:11:15.123634+06');
microsecond
-----
123634
(1 row)
```

minute(timestamp with time zone)

Description: Returns the minute value in the time.

Return type: integer

Example:

```
SELECT minute(timestampz '2018-12-13 12:11:15+06');
minute
-----
11
(1 row)
```

month(date)

Description: Returns the month of a given datetime.

Return type: integer

Example:

```
select month('2020-11-30');
month
-----
11
(1 row)
```

now([fsp])

Description: Returns the start time of the transaction. The parameter determines the microsecond output precision. The default value is 6.

Return type: timestamp with time zone

Example:

```
SELECT now();
now
-----
2017-09-01 17:03:42.549426+08
(1 row)
SELECT now(3);
now
-----
2021-09-08 10:59:00.427+08
(1 row)
```

numtodsinterval(num, interval_unit)

Description: Converts a number to an interval. **num** is a numeric-typed number. **interval_unit** is a string in the following format: 'DAY' | 'HOUR' | 'MINUTE' | 'SECOND'

You can set the **IntervalStyle** parameter to **oracle** to be compatible with the interval output format of the function in the Oracle database.

Example:

```
SELECT numtodsinterval(100, 'HOUR');
numtodsinterval
-----
100:00:00
(1 row)

SET intervalstyle = oracle;
SET
SELECT numtodsinterval(100, 'HOUR');
numtodsinterval
-----
+0000000004 04:00:00.000000000
(1 row)
```

pg_sleep(seconds)

Description: Returns the delay of the server thread in seconds.

Return type: void

Example:

```
SELECT pg_sleep(10);
pg_sleep
```

```
-----  
(1 row)
```

period_add(P, N)

Description: Returns the date of a given period plus *N* months.

Return type: integer

Example:

```
select period_add(200801, 2);  
period_add  
-----  
200803  
(1 row)
```

period_diff(P1, P2)

Description: Returns the number of months between two given dates.

Return type: integer

```
select period_diff(200802, 200703);  
period_diff  
-----  
11  
(1 row)
```

quarter(date)

Description: Returns the quarter to which the date belongs.

Return type: integer

Example:

```
SELECT quarter(date '2018-12-13');  
quarter  
-----  
4  
(1 row)
```

second(timestamp with time zone)

Description: Returns the second in a timestamp.

Return type: integer

Example:

```
SELECT second(timestampz '2018-12-13 12:11:15+06');  
second  
-----  
15  
(1 row)
```

statement_timestamp()

Description: Returns the current date and time.

Return type: timestamp with time zone

Example:

```
SELECT statement_timestamp();
       statement_timestamp
-----
2017-09-01 17:04:39.119267+08
(1 row)
```

sysdate

Description: Returns the current date and time.

Return type: timestamp

Example:

```
SELECT sysdate;
       sysdate
-----
2017-09-01 17:04:49
(1 row)
```

timeofday()

Description: Current date and time (like **clock_timestamp**, but returned as a **text** string)

Return type: text

Example:

```
SELECT timeofday();
       timeofday
-----
Fri Sep 01 17:05:01.167506 2017 CST
(1 row)
```

transaction_timestamp()

Description: Returns the current date and time. This function is equivalent to **current_timestamp**.

Return type: timestamp with time zone

Example:

```
SELECT transaction_timestamp();
       transaction_timestamp
-----
2017-09-01 17:05:13.534454+08
(1 row)
```

from_unixtime(unix_timestamp[,format])

Description: Converts a Unix timestamp to the datetime type when the format string is set to the default value. If the format string is specified, this function converts the Unix timestamp to a string of a specified format.

Return type: timestamp (default format string) or text (specified format string)

Example:

```
SELECT from_unixtime(875996580);
       from_unixtime
-----
```

```
-----  
1997-10-04 20:23:00  
(1 row)  
SELECT from_unixtime(875996580, '%Y %D %M %h:%i:%s');  
       from_unixtime  
-----  
1997 5th October 04:23:00  
(1 row)
```

unix_timestamp([timestamp with time zone])

Description: Returns the number of seconds from '1970-01-01 00:00:00'UTC to the timestamp specified by the input parameter. If no parameter is input, this function obtains the number of seconds from '1970-01-01 00:00:00'UTC to the current time.

Return type: bigint (no parameter is input) or numeric (parameter is input)

Example:

```
SELECT unix_timestamp();  
unix_timestamp  
-----  
1693906219  
(1 row)  
SELECT unix_timestamp('2018-09-08 12:11:13+06');  
unix_timestamp  
-----  
1536387073.000000  
(1 row)
```

add_months(d,n)

Description: Adds a specified number of months to a date.

Return type: timestamp

Example:

```
SELECT add_months(to_date('2017-5-29', 'yyyy-mm-dd'), 11) FROM dual;  
add_months  
-----  
2018-04-29 00:00:00  
(1 row)
```

last_day(d)

Description: Returns the last day of the month of a specified date.

- In the ORA- or TD-compatible mode, a timestamp is returned.
- In the MySQL-compatible mode, a date is returned.

Example:

```
select last_day(to_date('2017-01-01', 'YYYY-MM-DD')) AS cal_result;  
cal_result  
-----  
2017-01-31 00:00:00  
(1 row)
```

next_day(x,y)

Description: Returns a specified day of the next week of a given date.

- In the ORA- or TD-compatible mode, a timestamp is returned.
- In the MySQL-compatible mode, a date is returned.

Example:

```
select next_day(timestamp '2017-05-25 00:00:00','Sunday')AS cal_result;
       cal_result
-----
2017-05-28 00:00:00
(1 row)
```

from_days(days)

Description: Returns the corresponding date value based on the given number of days.

Return type: date

Example:

```
select from_days(730669);
       from_days
-----
2000-07-03
(1 row)
```

to_days(timestamp)

Description: Returns the number of days from the first day of year 0 to a specified date.

Return type: integer

Example:

```
SELECT to_days(timestamp '2008-10-07');
       to_days
-----
       733687
(1 row)
```

21.7.5.3 EXTRACT

EXTRACT(*field* FROM *source*)

The **extract** function retrieves subcolumns such as year or hour from date/time values. **source** must be a value expression of type **timestamp**, **time**, or **interval**. (Expressions of type **date** are cast to **timestamp** and can therefore be used as well.) **field** is an identifier or string that selects what column to extract from the source value. The **extract** function returns values of type **double precision**. The following are valid field names:

century

Century

The first century starts at 0001-01-01 00:00:00 AD. This definition applies to all Gregorian calendar countries. There is no century number 0. You go from **-1** century to **1** century.

Example:

```
SELECT EXTRACT(CENTURY FROM TIMESTAMP '2000-12-16 12:21:13');
date_part
-----
      20
(1 row)
```

day

- For **timestamp** values, the day (of the month) column (1–31)

```
SELECT EXTRACT(DAY FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
      16
(1 row)
```

- For **interval** values, the number of days

```
SELECT EXTRACT(DAY FROM INTERVAL '40 days 1 minute');
date_part
-----
      40
(1 row)
```

decade

Year column divided by 10

```
SELECT EXTRACT(DECADE FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
     200
(1 row)
```

dow

Day of the week as Sunday(0) to Saturday (6)

```
SELECT EXTRACT(DOW FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
       5
(1 row)
```

doy

Day of the year (1–365 or 366)

```
SELECT EXTRACT(DOY FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
      47
(1 row)
```

epoch

- For **timestamp with time zone** values, the number of seconds since 1970-01-01 00:00:00 UTC (can be negative);
for **date** and **timestamp** values, the number of seconds since 1970-01-01 00:00:00 local time;
for **interval** values, the total number of seconds in the interval.

```
SELECT EXTRACT(EPOCH FROM TIMESTAMP WITH TIME ZONE '2001-02-16 20:38:40.12-08');
date_part
-----
```

```
982384720.12
(1 row)
SELECT EXTRACT(EPOCH FROM INTERVAL '5 days 3 hours');
date_part
-----
442800
(1 row)
```

- Way to convert an epoch value back to a timestamp

```
SELECT TIMESTAMP WITH TIME ZONE 'epoch' + 982384720.12 * INTERVAL '1 second' AS RESULT;
result
-----
2001-02-17 12:38:40.12+08
(1 row)
```

hour

Hour column (0–23)

```
SELECT EXTRACT(HOUR FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
20
(1 row)
```

isodow

Day of the week (1–7)

Monday is 1 and Sunday is 7.

NOTE

This is identical to **dow** except for Sunday.

```
SELECT EXTRACT(ISODOW FROM TIMESTAMP '2001-02-18 20:38:40');
date_part
-----
7
(1 row)
```

isoyear

The ISO 8601 year that the date falls in (not applicable to intervals).

Each ISO year begins with the Monday of the week containing the 4th of January, so in early January or late December the ISO year may be different from the Gregorian year. For details, see [week](#).

```
SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-01');
date_part
-----
2005
(1 row)
SELECT EXTRACT(ISOYEAR FROM DATE '2006-01-02');
date_part
-----
2006
(1 row)
```

microseconds

The seconds column, including fractional parts, multiplied by 1,000,000

```
SELECT EXTRACT(MICROSECONDS FROM TIME '17:12:28.5');
date_part
-----
28500000
(1 row)
```

millennium

Millennium

Years in the 1900s are in the second millennium. The third millennium started from January 1, 2001.

```
SELECT EXTRACT(MILLENNIUM FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
3
(1 row)
```

milliseconds

The seconds column, including fractional parts, multiplied by 1000. Note that this includes full seconds.

```
SELECT EXTRACT(MILLISECONDS FROM TIME '17:12:28.5');
date_part
-----
28500
(1 row)
```

minute

Minutes column (0–59)

```
SELECT EXTRACT(MINUTE FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
38
(1 row)
```

month

For **timestamp** values, the number of the month within the year (1–12);

```
SELECT EXTRACT(MONTH FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
2
(1 row)
```

For **interval** values, the number of months, modulo 12 (0–11)

```
SELECT EXTRACT(MONTH FROM INTERVAL '2 years 13 months');
date_part
-----
1
(1 row)
```

quarter

Quarter of the year (1–4) that the date is in

```
SELECT EXTRACT(QUARTER FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
1
(1 row)
```



```
-----  
      1  
(1 row)
```

second

Seconds column, including fractional parts (0–59)

```
SELECT EXTRACT(SECOND FROM TIME '17:12:28.5');  
date_part  
-----  
      28.5  
(1 row)
```

timezone

The time zone offset from UTC, measured in seconds. Positive values correspond to time zones east of UTC, negative values to zones west of UTC.

```
SELECT EXTRACT(timezone FROM TIMETZ '17:12:28');  
date_part  
-----  
     28800  
(1 row)
```

timezone_hour

The hour component of the time zone offset

```
SELECT EXTRACT(timezone_hour FROM TIMETZ '17:12:28');  
date_part  
-----  
      8  
(1 row)
```

timezone_minute

The minute component of the time zone offset

```
SELECT EXTRACT(timezone_minute FROM TIMETZ '17:12:28');  
date_part  
-----  
      0  
(1 row)
```

week

The number of the week of the year that the day is in. By definition (ISO 8601), the first week of a year contains January 4 of that year. (The ISO-8601 week starts on Monday.) In other words, the first Thursday of a year is in week 1 of that year.

Because of this, it is possible for early January dates to be part of the 52nd or 53rd week of the previous year, and late December dates to be part of the 1st week of the next year. For example, **2005-01-01** is part of the 53rd week of year 2004, **2006-01-01** is part of the 52nd week of year 2005, and **2012-12-31** is part of the 1st week of year 2013. You are advised to use the columns **isoyear** and **week** together to ensure consistency.

```
SELECT EXTRACT(WEEK FROM TIMESTAMP '2001-02-16 20:38:40');  
date_part  
-----
```

```
7
(1 row)
```

year

Year column

```
SELECT EXTRACT(YEAR FROM TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
2001
(1 row)
```

21.7.5.4 date_part

date_part

The **date_part** function is modeled on the traditional Ingres equivalent to the SQL-standard function **extract**:

`date_part('field', source)`

Note that the **field** must be a string, rather than a name. The valid field names are the same as those for **extract**. For details, see [EXTRACT](#).

Example:

```
SELECT date_part('day', TIMESTAMP '2001-02-16 20:38:40');
date_part
-----
16
(1 row)
SELECT date_part('hour', INTERVAL '4 hours 3 minutes');
date_part
-----
4
(1 row)
```

21.7.5.5 date_format

date_format(timestamp, fmt)

Converts a date into a string in the format specified by **fmt**.

Example:

```
SELECT date_format('2009-10-04 22:23:00', '%M %D %W');
date_format
-----
October 4th Sunday
(1 row)
SELECT date_format('2021-02-20 08:30:45', '%Y-%m-%d %H:%i:%S');
date_format
-----
2021-02-20 08:30:45
(1 row)
SELECT date_format('2021-02-20 18:10:15', '%r-%T');
date_format
-----
06:10:15 PM-18:10:15
(1 row)
```

Table 21-33 describes the patterns of date parameter values. They can be used for the **date_format**, **time_format**, **str_to_date**, **str_to_time**, and **from_unixtime** functions.

Table 21-33 Formats for the output string

| Format | Description | Value |
|--------|---|-------------------------|
| %a | Abbreviated week name | Sun...Sat |
| %b | Abbreviated month name | Jan...Dec |
| %c | Month | 0...12 |
| %D | Date with a suffix | 0th, 1st, 2nd, 3rd, ... |
| %d | Day in a month (two digits) | 00...31 |
| %e | Day in a month | 0...31 |
| %f | Microsecond | 000000...999999 |
| %H | Hour, in 24-hour format | 00...23 |
| %h | Hour, in 12-hour format | 01...12 |
| %l | Hour, in 12-hour format, same as %h | 01...12 |
| %i | Minute | 00...59 |
| %j | Day in a year | 001...366 |
| %k | Hour, in 24-hour format, same as %H | 0...23 |
| %l | Hour, in 12-hour format, same as %h | 1...12 |
| %M | Month name | January...December |
| %m | Month (two digits) | 00...12 |
| %p | Morning and afternoon | AM PM |
| %r | Time, in 12-hour format | hh:mm:ss AM/PM |
| %S | Second | 00...59 |
| %s | Second, same as %S | 00...59 |
| %T | Time, in 24-hour format | hh:mm:ss |
| %U | Week (Sunday is the first day of a week.) | 00...53 |
| %u | Week (Monday is the first day of a week.) | 00...53 |

| Format | Description | Value |
|--------|---|-------------------|
| %V | Week (Sunday is the first day of a week). It is used together with %X. | 01...53 |
| %v | Week (Monday is the first day of a week). It is used together with %x. | 01...53 |
| %W | Week name | Sunday...Saturday |
| %w | Day of a week. The value is 0 for Sunday. | 0...6 |
| %X | Year (four digits). It is used together with %V. Sunday is the first day of a week. | - |
| %x | Year (four digits). It is used together with %v. Monday is the first day of a week. | - |
| %Y | Year (four digits) | - |
| %y | Year (two digits) | - |
| %% | Character '%' | Character '%' |
| %x | 'x': any character apart from the preceding ones | Character 'x' |

NOTICE

In the preceding table, %U, %u, %V, %v, %X, and %x are not supported currently.

21.7.5.6 time_format

time_format(time, fmt)

Description: The **time_format** function converts the **date** parameter into a string in the format specified by **fmt**. It is similar to the **date_format** function, but the format string can contain only hour, minute, second, and microsecond format specifiers. If other specifiers are contained, **NULL** or **0** is returned.

Return type: text

Example:

```
SELECT time_format('2009-10-04 22:23:00', '%M %D %W');
time_format
-----
(1 row)
SELECT time_format('2021-02-20 08:30:45', '%Y-%m-%d %H:%i:%S');
```

```
time_format
-----
0000-00-00 08:30:45
(1 row)
SELECT time_format('2021-02-20 18:10:15', '%r-%T');
time_format
-----
06:10:15 PM-18:10:15
(1 row)
```

NOTICE

time_format supports only time-related formats (%f, %H, %h, %l, %i, %k, %l, %p, %r, %S, %s, and %T) and does not support date-related formats. In other cases, **time_format** is output as common characters.

str_to_date(str, format)

Description: Converts a string of the date/time type to a value of the date type according to the provided display format.

Return type: timestamp

Example:

```
SELECT str_to_date('01,5,2021','%d,%m,%Y');
str_to_date
-----
2021-05-01 00:00:00
(1 row)
SELECT str_to_date('01,5,2021,09,30,17','%d,%m,%Y,%h,%i,%s');
str_to_date
-----
2021-05-01 09:30:17
(1 row)
```

For details about the input format types applicable to **str_to_date**, see [Table 21-33](#). Only input values of the date or date/time type can be converted. Use **str_to_time** when only values of the time type are input.

str_to_time(str, format)

Description: Converts a string of the time type to a value of the time type according to the provided display format.

Return type: time

Example:

```
SELECT str_to_time('09:30:17','%h:%i:%s');
str_to_time
-----
09:30:17
(1 row)
```

For details about the input format types applicable to **str_to_time**, see [Table 21-33](#). Only input values of the time type can be converted. Use **str_to_date** when values of the date or date/time type are input.

week(date[, mode])

Description: Returns the number of weeks in the year of the specified datetime.
The default value is **0**.

Return type: integer

Table 21-34 Working principle of the mode in the week function

| Sche
ma | First Day of a
Week | Week Range | Rule for Determining the First Week |
|------------|------------------------|------------|--|
| 0 | Sun | 0-53 | Week of the first Sunday after New Year's Day |
| 1 | Mon | 0-53 | Week with four or more days after New Year's Day |
| 2 | Sun | 1-53 | Week of the first Sunday after New Year's Day |
| 3 | Mon | 1-53 | Week with four or more days after New Year's Day |
| 4 | Sun | 0-53 | Week with four or more days after New Year's Day |
| 5 | Mon | 0-53 | Week of the first Monday after New Year's Day |
| 6 | Sun | 1-53 | Week with four or more days after New Year's Day |
| 7 | Mon | 1-53 | Week of the first Monday after New Year's Day |

Example:

```
select week('2018-01-01');
week
-----
0
(1 row)

select week('2018-01-01', 0);
week
-----
0
(1 row)

select week('2020-12-31', 1);
week
-----
53
(1 row)

select week('2020-12-31', 5);
week
-----
```

```
52
(1 row)
```

weekday(date)

Description: Returns the week index corresponding to the given date, with Monday as the start day of the week.

Value range: 0 to 6

Return type: integer

Example:

```
select weekday('2020-11-06');
weekday
-----
4
(1 row)
```

weekofyear(date)

Description: Returns the number of weeks in the current year for the week of the given date. The value ranges from 1 to 53, which is equivalent to week(date, 3).

Return type: integer

Example:

```
select weekofyear('2020-12-30');
weekofyear
-----
53
(1 row)
```

year(date)

Description: Obtains the year of the **date**.

Return type: integer

Example:

```
select year('2020-11-13');
year
-----
2020
(1 row)
```

yearweek(date[, mode])

Description: Returns the year and the number of weeks in the current year for the given date. The number of weeks ranges from 1 to 53.

Return type: integer

Example:

```
select yearweek('2019-12-31');
yearweek
-----
201952
(1 row)
```

```
select yearweek('2019-1-1');
yearweek
-----
201852
(1 row)
```

21.7.6 SEQUENCE Functions

The sequence functions provide a simple method to ensure security of multiple users for users to obtain sequence values from sequence objects.

nextval(regclass)

Specifies an increasing sequence and returns a new value.

NOTE

- In order to avoid blocking concurrent transactions that retrieve values from the same sequence, the **nextval** operation does not undo its action. This means that once a value is obtained, it is considered as used and will not be given again. This applies even if the operation is part of a transaction that gets interrupted or if the calling query does not end up using the value. This means that aborted transactions may leave unused "holes" in the sequence of assigned values. Therefore, sequences in DWS cannot be used to obtain sequence without gaps.
- If the **nextval** function is pushed to DN, each DN will automatically connect to the GTM and requests the next value. For example, in the **INSERT INTO t1 SELECT xxx** statement, a column in table **t1** needs to invoke the **nextval** function. If maximum number of connections on the GTM is 8,192, this type of pushed statements occupies too many GTM connections. Therefore, the number of concurrent connections for these statements is limited to 7,000 divided by the number of cluster DNs. The other 1,192 connections are reserved for other statements.

Return type: bigint

The **nextval** function can be invoked in either of the following ways: (In example 2, the Oracle syntax is supported. Currently, the sequence name cannot contain a dot.)

Example 1:

```
SELECT nextval('seqDemo');
nextval
-----
2
(1 row)
```

Example 2:

```
SELECT seqDemo.nextval;
nextval
-----
2
(1 row)
```

currval(regclass)

Returns the last value of **nextval** for a specified sequence in the current session. If **nextval** has not been invoked for the specified sequence in the current session, an error is reported when **currval** is invoked. By default, **currval** is disabled. To enable it, set **enable_beta_features** to **true**. After **currval** is enabled, **nextval** will not be pushed down.

Return type: bigint

The **currval** function can be invoked in either of the following ways: (In example 2, the Oracle syntax is supported. Currently, the sequence name cannot contain a dot.)

Example 1:

```
SELECT currval('seq1');
currval
-----
      2
(1 row)
```

Example 2:

```
SELECT seq1.currval seq1;
currval
-----
      2
(1 row)
```

lastval()

Returns the last value of **nextval** in the current session. This function is equivalent to **currval**, but **lastval** does not have a parameter. If **nextval** has not been invoked in the current session, an error is reported when **lastval** is invoked.

By default, **lastval** is disabled. To enable it, set **enable_beta_features** or **lastval_supported** to **true**. After **lastval** is enabled, **nextval** will not be pushed down.

Return type: bigint

For example:

```
SELECT lastval();
lastval
-----
      2
(1 row)
```

setval(regclass, bigint)

Sets the current value of a sequence.

Return type: bigint

For example:

```
SELECT setval('seqDemo',1);
setval
-----
      1
(1 row)
```

setval(regclass, bigint, boolean)

Sets the current value of a sequence and the `is_called` sign.

Return type: bigint

For example:

```
SELECT setval('seqDemo',1,true);
setval
-----
1
(1 row)
```

NOTE

The current session and GTM will take effect immediately after **setval** is performed. If other sessions have buffered sequence values, **setval** will take effect only after the values are used up. Therefore, to prevent sequence value conflicts, you are advised to use **setval** with caution.

Because the sequence is non-transactional, changes made by **setval** will not be canceled when a transaction rolled back.

21.7.7 Array Functions and Operators

21.7.7.1 Array Operators

Array comparisons compare the array contents element-by-element, using the default B-tree comparison function for the element data type. In multidimensional arrays, the elements are accessed in row-major order. If the contents of two arrays are equal but the dimensionality is different, the first difference in the dimensionality information determines the sort order.

=

Description: Specifies whether two arrays are equal.

Example:

```
SELECT ARRAY[1,1,2,1,3,1]::int[] = ARRAY[1,2,3] AS RESULT;
result
-----
t
(1 row)
```

<>

Description: Specifies whether two arrays are not equal.

Example:

```
SELECT ARRAY[1,2,3] <> ARRAY[1,2,4] AS RESULT;
result
-----
t
(1 row)
```

<

Description: Specifies whether an array is less than another.

Example:

```
SELECT ARRAY[1,2,3] < ARRAY[1,2,4] AS RESULT;
result
-----
t
(1 row)
```

>

Description: Specifies whether an array is greater than another.

Example:

```
SELECT ARRAY[1,4,3] > ARRAY[1,2,4] AS RESULT;  
result  
-----  
t  
(1 row)
```

<=

Description: Specifies whether an array is less than another.

Example:

```
SELECT ARRAY[1,2,3] <= ARRAY[1,2,3] AS RESULT;  
result  
-----  
t  
(1 row)
```

>=

Description: Specifies whether an array is greater than or equal to another.

Example:

```
SELECT ARRAY[1,4,3] >= ARRAY[1,4,3] AS RESULT;  
result  
-----  
t  
(1 row)
```

@>

Description: Specifies whether an array contains another.

Example:

```
SELECT ARRAY[1,4,3] @> ARRAY[3,1] AS RESULT;  
result  
-----  
t  
(1 row)
```

<@

Description: Specifies whether an array is contained in another.

Example:

```
SELECT ARRAY[2,7] <@ ARRAY[1,7,4,2,6] AS RESULT;  
result  
-----  
t  
(1 row)
```

&&

Description: Specifies whether an array overlaps another (have common elements).

Example:

```
SELECT ARRAY[1,4,3] && ARRAY[2,1] AS RESULT;  
result  
-----  
t  
(1 row)
```

||

Description: Array-to-array concatenation

Example:

```
SELECT ARRAY[1,2,3] || ARRAY[4,5,6] AS RESULT;  
result  
-----  
{1,2,3,4,5,6}  
(1 row)  
SELECT ARRAY[1,2,3] || ARRAY[[4,5,6],[7,8,9]] AS RESULT;  
result  
-----  
{{1,2,3},{4,5,6},{7,8,9}}  
(1 row)
```

||

Description: Element-to-array concatenation

Example:

```
SELECT 3 || ARRAY[4,5,6] AS RESULT;  
result  
-----  
{3,4,5,6}  
(1 row)
```

||

Description: Array-to-element concatenation

Example:

```
SELECT ARRAY[4,5,6] || 7 AS RESULT;  
result  
-----  
{4,5,6,7}  
(1 row)
```

21.7.7.2 Array Functions

array_append(anyarray, anyelement)

Description: Appends an element to the end of an array, and only supports dimension-1 arrays.

Return type: anyarray

Example:

```
SELECT array_append(ARRAY[1,2], 3) AS RESULT;  
result  
-----  
{1,2,3}  
(1 row)
```

array_prepend(anelement, anyarray)

Description: Appends an element to the beginning of an array, and only supports dimension-1 arrays.

Return type: anyarray

Example:

```
SELECT array_prepend(1, ARRAY[2,3]) AS RESULT;  
result  
-----  
{1,2,3}  
(1 row)
```

array_cat(anyarray, anyarray)

Description: Concatenates two arrays, and supports multi-dimensional arrays.

Return type: anyarray

Example:

```
SELECT array_cat(ARRAY[1,2,3], ARRAY[4,5]) AS RESULT;  
result  
-----  
{1,2,3,4,5}  
(1 row)  
  
SELECT array_cat(ARRAY[[1,2],[4,5]], ARRAY[6,7]) AS RESULT;  
result  
-----  
{{1,2},{4,5},{6,7}}  
(1 row)
```

array_ndims(anyarray)

Description: Returns the number of dimensions of the array.

Return type: int

Example:

```
SELECT array_ndims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;  
result  
-----  
2  
(1 row)
```

array_dims(anyarray)

Description: Returns a text representation of array's dimensions.

Return type: text

Example:

```
SELECT array_dims(ARRAY[[1,2,3], [4,5,6]]) AS RESULT;  
result  
-----  
[1:2][1:3]  
(1 row)
```

array_length(anyarray, int)

Description: Returns the length of the requested array dimension.

Return type: int

Example:

```
SELECT array_length(array[1,2,3], 1) AS RESULT;  
result  
-----  
3  
(1 row)
```

array_lower(anyarray, int)

Description: Returns lower bound of the requested array dimension.

Return type: int

Example:

```
SELECT array_lower('[0:2]={1,2,3}'::int[], 1) AS RESULT;  
result  
-----  
0  
(1 row)
```

array_upper(anyarray, int)

Description: Returns upper bound of the requested array dimension.

Return type: int

Example:

```
SELECT array_upper(ARRAY[1,8,3,7], 1) AS RESULT;  
result  
-----  
4  
(1 row)
```

array_to_string(anyarray, text [, text])

Description: Uses the first **text** as the new delimiter and the second **text** to replace **NULL** values.

Return type: text

Example:

```
SELECT array_to_string(ARRAY[1, 2, 3, NULL, 5], ',', '*') AS RESULT;  
result  
-----  
1,2,3*,5  
(1 row)
```

 NOTE

In **string_to_array**, if the null-string parameter is omitted or NULL, none of the substrings of the input will be replaced by NULL.

In **array_to_string**, if the null-string parameter is omitted or NULL, any null elements in the array are simply skipped and not represented in the output string.

string_to_array(text, text [, text])

Description: Uses the second **text** as the new delimiter and the third **text** as the substring to be replaced by **NULL** values. A substring can be replaced by **NULL** values only when it is the same as the third **text**.

Return type: text[]

Example:

```
SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'yy') AS RESULT;
result
-----
{xx,NULL,zz}
(1 row)
SELECT string_to_array('xx~^~yy~^~zz', '~^~', 'y') AS RESULT;
result
-----
{xx,yy,zz}
(1 row)
```

 NOTE

In **string_to_array**, if the delimiter parameter is NULL, each character in the input string will become a separate element in the resulting array. If the delimiter is an empty string, then the entire input string is returned as a one-element array. Otherwise the input string is split at each occurrence of the delimiter string.

unnest(anyarray)

Description: Expands an array to a set of rows.

Return type: setof anyelement

Example:

```
SELECT unnest(ARRAY[1,2]) AS RESULT;
result
-----
1
2
(2 rows)
```

The **unnest** function is used together with the **string_to_array** array. To convert an array to columns, the statement first splits a string into arrays by comma, and then converts the arrays into columns.

```
SELECT unnest(string_to_array('a,b,c,d',';')) AS RESULT;
result
-----
a
b
c
d
(4 rows)
```

interval(N, N1, N2, N3 ...)

Description: Searches for the last array index that is less than or equal to the target parameter **n** from the input integer array. If **n** is NULL, -1 is returned. The **interval()** function does not support the interval(N, N1) scenario. This function is supported by version 8.2.0 or later clusters.

Return type: int

Example:

```
SELECT INTERVAL(10, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 11) AS RESULT;  
result  
-----  
11  
(1 row)
```

split(string, delim)

Description: Separates strings by a delimiter and returns an array. This function is supported by version 8.2.0 or later clusters.

Return type: text[]

Example:

```
SELECT SPLIT('a-b-c-d-e', '-') AS RESULT;  
result  
-----  
{a,b,c,d,e}  
(1 row)  
SELECT SPLIT('a-b-c-d-e', '-')[4] AS RESULT;  
result  
-----  
d  
(1 row)
```

21.7.8 Logical Operators

The usual logical operators include AND, OR, and NOT. SQL uses a three-valued logical system with true, false, and null, which represents "unknown". Their priorities are NOT > AND > OR.

Table 21-35 lists operation rules, where **a** and **b** represent logical expressions.

Table 21-35 Operation rules

| a | b | a AND b Result | a OR b Result | NOT a Result |
|-------|-------|----------------|---------------|--------------|
| TRUE | TRUE | TRUE | TRUE | FALSE |
| TRUE | FALSE | FALSE | TRUE | FALSE |
| TRUE | NULL | NULL | TRUE | FALSE |
| FALSE | FALSE | FALSE | FALSE | TRUE |
| FALSE | NULL | FALSE | NULL | TRUE |
| NULL | NULL | NULL | NULL | NULL |

 NOTE

The operators AND and OR are commutative, that is, you can switch the left and right operand without affecting the result.

21.7.9 Comparison Operators

Comparison operators can be used for all data types. The return results are Boolean values.

All comparison operators are binary operators. Only data types that are the same or can be implicitly converted can be compared using comparison operators. Expressions such as "1<2<3" are invalid because Boolean values cannot be compared with 3.

Table 21-36 describes comparison operators provided by DWS.

Table 21-36 Comparison operators

| Operators | Description |
|-----------|--------------------------|
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| = | Equality |
| <> or != | Inequality |

21.7.10 Pattern Matching Operators

There are three separate approaches to pattern matching provided by the database: the traditional SQL LIKE operator, the more recent SIMILAR TO operator, and POSIX-style regular expressions. Besides these basic operators, functions can be used to extract or replace matching substrings and to split a string at matching locations.

LIKE

Checks whether the string matches the pattern string following **LIKE**. If the string matches the supplied pattern, the LIKE expression returns true (the NOT LIKE expression returns false). Otherwise, the LIKE expression returns false (the NOT LIKE expression returns true).

- Rule
 - a. This operator can succeed only when its pattern matches the entire string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.

- b. The underscore (_) represents (matching) any single character. Percentage (%) indicates the wildcard character of any string.
- c. To match a literal underscore (_) or percent sign (%) without matching other characters, the respective character in pattern must be preceded by the escape character. The default escape character is the backslash but a different one can be selected by using the ESCAPE clause.
- d. To match the escape character itself, write two escape characters. For example: To write a **pattern** constant containing a backslash (\), you need to enter two backslashes in SQL statements.

NOTE

When **standard_conforming_strings** is set to **off**, any backslashes you write in literal string constants will need to be doubled. Therefore, writing a pattern matching a single backslash is actually going to write four backslashes in the statement. You can avoid this by selecting a different escape character by using ESCAPE, so that the backslash is no longer a special character of LIKE. But the backslash is still the special character of the character text analyzer, so you still need two backslashes. You can also select no escape character by writing **ESCAPE ''**. This effectively disables the escape mechanism, which makes it impossible to turn off the special meaning of underscore and percent signs in the pattern.

- e. The keyword ILIKE can be used instead of LIKE to make the match case-insensitive.
- f. Operator ~~ is equivalent to LIKE, and operator ~~* corresponds to ILIKE.

- Examples

```
SELECT 'abc' LIKE 'abc' AS RESULT;
result
-----
t
(1 row)
SELECT 'abc' LIKE 'a%' AS RESULT;
result
-----
t
(1 row)
SELECT 'abc' LIKE '_b_' AS RESULT;
result
-----
t
(1 row)
SELECT 'abc' LIKE 'c' AS RESULT;
result
-----
f
(1 row)
```

SIMILAR TO

The **SIMILAR TO** operator determines whether to match a given string based on its own pattern and returns **true** or **false**. It is similar to LIKE, except that it interprets the pattern using the SQL standard's definition of a regular expression.

- Matching rule
 - a. Like LIKE, the SIMILAR TO operator returns true only if its pattern matches the given string. If you want to match a sequence in any position within the string, the pattern must begin and end with a percent sign.

- b. The underscore (_) represents (matching) any single character. Percentage (%) indicates the wildcard character of any string.
- c. SIMILAR TO supports these pattern-matching metacharacters borrowed from POSIX regular expressions:

Table 21-37 Pattern matching metacharacter

| Metacharacter | Description |
|---------------|---|
| | Specifies alternation (either of two alternatives). |
| * | Specifies repetition of the previous item zero or more times. |
| + | Specifies repetition of the previous item one or more times. |
| ? | Specifies repetition of the previous item zero or one time. |
| {m} | Specifies repetition of the previous item exactly <i>m</i> times. |
| {m,} | Specifies repetition of the previous item <i>m</i> or more times. |
| {m,n} | Specifies repetition of the previous item at least <i>m</i> times and does not exceed <i>n</i> times. |
| () | Specifies that parentheses () can be used to group items into a single logical item. |
| [...] | Specifies a character class, just as in POSIX regular expressions. |

- d. A preamble escape character disables the special meaning of any of these metacharacters. The rules for using escape characters are the same as those for LIKE.
- Precautions

If a large number of characters are repeatedly matched in the SIMILAR TO regular expression, the statement fails to be executed and error "invalid regular expression: regular expression is too complex" is reported due to the recursion size restriction. In this case, increase the value of max_stack_depth.
- Regular expression functions

The **substring(string from pattern for escape)** function can be used to intercept a substring that matches an SQL regular expression.
- Examples

```
SELECT 'abc' SIMILAR TO 'abc' AS RESULT;
result
-----
t
(1 row)
```

```
SELECT 'abc' SIMILAR TO 'a' AS RESULT;
result
-----
f
(1 row)
SELECT 'abc' SIMILAR TO '%(b|d)%' AS RESULT;
result
-----
t
(1 row)
SELECT 'abc' SIMILAR TO '(b|c)%' AS RESULT;
result
-----
f
(1 row)
```

POSIX regular expressions

A regular expression is a character sequence that is an abbreviated definition of a set of strings (a regular set). If a string is a member of a regular expression described by a regular expression, the string matches the regular expression. POSIX regular expressions provide a more powerful means for pattern matching than the LIKE and SIMILAR TO operators. [Table 21-38](#) lists all available operators for POSIX regular expression pattern matching.

Table 21-38 Regular expression match operators

| Operator | Description | Example |
|----------|---|--------------------------|
| ~ | Matches regular expression, which is case-sensitive. | 'thomas' ~ '.*thomas.*' |
| ~* | Matches regular expression, which is case-insensitive. | 'thomas' ~* '.*Thomas.*' |
| !~ | Does not match regular expression, which is case-sensitive. | 'thomas' !~ '.*Thomas.*' |
| !~* | Does not match regular expression, which is case-insensitive. | 'thomas' !~* '.*vadim.*' |

- Matching rule
 - Unlike LIKE patterns, a regular expression is allowed to match anywhere within a string, unless the regular expression is explicitly anchored to the beginning or end of the string.
 - Besides the metacharacters mentioned above, POSIX regular expressions also support the following pattern matching metacharacters:

Table 21-39 Pattern matching metacharacters

| Metacharacter | Description |
|---------------|---|
| ^ | Specifies the match starting with a string. |
| \$ | Specifies the match at the end of a string. |
| . | Matches any single character. |

- Regular expression functions

POSIX regular expressions support the following functions:

- The **substring(string from pattern)** function provides a method for extracting a substring that matches the POSIX regular expression pattern.
- The **regexp_replace(string, pattern, replacement [, flags])** function provides the function of replacing the substring matching the POSIX regular expression pattern with the new text.
- The **regexp_matches(string text, pattern text [, flags text])** function returns a text array consisting of all captured substrings that match a POSIX regular expression pattern.
- The **regexp_split_to_table(string text, pattern text [, flags text])** function splits a string using a POSIX regular expression pattern as a delimiter.
- The **regexp_split_to_array(string text, pattern text [, flags text])** function behaves the same as **regexp_split_to_table**, except that **regexp_split_to_array** returns its result as an array of text.

 NOTE

The regular expression split functions ignore zero-length matches, which occur at the beginning or end of a string or after the previous match. This is contrary to the strict definition of regular expression matching. The latter is implemented by **regexp_matches**, but the former is usually the most commonly used behavior in practice.

- Examples

```
SELECT 'abc' ~ 'Abc' AS RESULT;
result
-----
f
(1 row)
SELECT 'abc' ~* 'Abc' AS RESULT;
result
-----
t
(1 row)
SELECT 'abc' !~ 'Abc' AS RESULT;
result
-----
t
(1 row)
SELECT 'abc' !~* 'Abc' AS RESULT;
result
-----
f
(1 row)
```

```
SELECT 'abc' ~ '^a' AS RESULT;  
result  
-----  
t  
(1 row)  
SELECT 'abc' ~ '(b|d)' AS RESULT;  
result  
-----  
t  
(1 row)  
SELECT 'abc' ~ '^ (b|c)' AS RESULT;  
result  
-----  
f  
(1 row)
```

Although most regular expression searches can be executed quickly, the time and memory for regular expression processing can still be manually controlled. Avoid using regular expression search patterns from untrusted sources. If required, set a statement timeout limit. The search with the SIMILAR TO mode has the same security risks as the SIMILAR TO provides many capabilities that are the same as those of the POSIX- style regular expression. For better security with unreliable inputs, prefer the simpler LIKE search over complex alternatives.

21.7.11 Aggregate Functions

sum(expression)

Description: Sum of expression across all input values

Return type:

Generally, same as the argument data type. In the following cases, type conversion occurs:

- **BIGINT** for **SMALLINT** or **INT** arguments
- **NUMBER** for **BIGINT** arguments
- **DOUBLE PRECISION** for floating-point arguments

Example:

```
postgres=#SELECT SUM(ss_ext_tax) FROM tpeds.STORE_SALES;  
sum  
-----  
213267594.69  
(1 row)
```

max(expression)

Description: Specifies the maximum value of **expression** across all input values.

Argument types: any array, numeric, string, or date/time type

Return type: same as the argument type

Example:

```
postgres=#SELECT MAX(inv_quantity_on_hand) FROM tpeds.inventory;  
max  
-----
```

```
1000000  
(1 row)
```

min(expression)

Description: Specifies the minimum value of **expression** across all input values.

Argument types: any array, numeric, string, or date/time type

Return type: same as the argument type

Example:

```
postgres=#SELECT MIN(inv_quantity_on_hand) FROM tpceds.inventory;  
min  
-----  
0  
(1 row)
```

avg(expression)

Description: Average (arithmetic mean) of all input values

Return type:

- **NUMBER** for any integer-type argument.
- **DOUBLE PRECISION** for floating-point parameters.
- Other values are the same as the input data type.

Example:

```
postgres=#SELECT AVG(inv_quantity_on_hand) FROM tpceds.inventory;  
avg  
-----  
500.0387129084044604  
(1 row)
```

median(expression)

Description: Median of all input values Currently, only the numeric and interval types are supported. Null values are not used for calculation.

Return type:

- If all input values are integers, a median of the **NUMERIC** type is returned; otherwise, a median of the same type as the input values is returned.
- In the Teradata-compatible mode, if the input values are integers, the returned median is rounded to the nearest integer.

Example:

```
postgres=#SELECT MEDIAN(inv_quantity_on_hand) FROM tpceds.inventory;  
median  
-----  
500  
(1 row)
```

percentile_cont(const) within group(order by expression)

Description: returns a value corresponding to the specified percentile in the ordering, interpolating between adjacent input items if needed. Null values are not used for calculation.

Input: *const* indicates a number ranging from 0 to 1. Currently, only numeric and interval expressions are supported.

Return type:

- If all input values are integers, a median of the **NUMERIC** type is returned; otherwise, a median of the same type as the input values is returned.
- In the Teradata-compatible mode, if the input values are integers, the returned median is rounded to the nearest integer.

Example:

```
postgres=#SELECT percentile_cont(0.3) within group(order by x) FROM (SELECT generate_series(1,5) AS x) AS t;
percentile_cont
-----
2.2
(1 row)
postgres=#SELECT percentile_cont(0.3) within group(order by x desc) FROM (SELECT generate_series(1,5) AS x) AS t;
percentile_cont
-----
3.8
(1 row)
```

percentile_disc(const) within group(order by expression)

Description: returns the first input value whose position in the ordering equals or exceeds the specified percentile.

Input: *const* indicates a number ranging from 0 to 1. Currently, only numeric and interval expressions are supported. Null values are not used for calculation.

Return type: If all input values are integers, a median of the **NUMERIC** type is returned; otherwise, a median of the same type as the input values is returned.

Example:

```
postgres=#SELECT percentile_disc(0.3) within group(order by x) FROM (SELECT generate_series(1,5) AS x) AS t;
percentile_disc
-----
2
(1 row)
postgres=#SELECT percentile_disc(0.3) within group(order by x desc) FROM (SELECT generate_series(1,5) AS x) AS t;
percentile_disc
-----
4
(1 row)
```

count(expression)

Description: Number of input rows for which the value of expression is not null

Return type: bigint

Example:

```
postgres=#SELECT COUNT(inv_quantity_on_hand) FROM tpeds.inventory;
count
-----
11158087
(1 row)
```

count(*)

Description: Number of input rows

Return type: bigint

Example:

```
postgres=#SELECT COUNT(*) FROM tpeds.inventory;
count
-----
11745000
(1 row)
```

array_agg(expression)

Description: Input values, including nulls, concatenated into an array The input parameters of the function do not support the array format.

Return type: array of the argument type

Example:

Create the **employeeinfo** table and insert data into the table:

```
CREATE TABLE employeeinfo (empno smallint, ename varchar(20), job varchar(20), hiredate date,deptno smallint);
INSERT INTO employeeinfo VALUES (7155, 'JACK', 'SALESMAN', '2018-12-01', 30);
INSERT INTO employeeinfo VALUES (7003, 'TOM', 'FINANCE', '2016-06-15', 20);
INSERT INTO employeeinfo VALUES (7357, 'MAX', 'SALESMAN', '2020-10-01', 30);

SELECT * FROM employeeinfo;
 empno | ename | job   | hiredate       | deptno
-----+-----+-----+-----+-----
 7155 | JACK | SALESMAN | 2018-12-01 00:00:00 |    30
 7357 | MAX  | SALESMAN | 2020-10-01 00:00:00 |    30
 7003 | TOM  | FINANCE  | 2016-06-15 00:00:00 |    20
(3 rows)
```

Query the names of all employees in the department whose ID is **30**:

```
SELECT array_agg(ename) FROM employeeinfo where deptno = 30;
array_agg
-----
{JACK,MAX}
(1 row)
```

Query all employees in the same department:

```
SELECT deptno, array_agg(ename) FROM employeeinfo group by deptno;
deptno | array_agg
-----+-----
    30 | {JACK,MAX}
    20 | {TOM}
(2 rows)

SELECT distinct array_agg(ename) OVER (PARTITION BY deptno) FROM employeeinfo;
array_agg
-----
```

```
{TOM}  
{JACK,MAX}  
(2 rows)
```

Query all department IDs and deduplicate them:

```
SELECT array_agg(distinct deptno) FROM employeeinfo group by deptno;  
array_agg  
-----  
{20}  
{30}  
(2 rows)
```

Sort the deduplicated department IDs in descending order:

```
SELECT array_agg(distinct deptno order by deptno desc) FROM employeeinfo;  
array_agg  
-----  
{30,20}  
(1 row)
```

string_agg(expression, delimiter)

Description: Input values concatenated into a string, separated by delimiter

Return type: same as the argument type

Example:

Query all employees in the same department based on the created table **employeeinfo**:

```
SELECT deptno, string_agg(ename,')') FROM employeeinfo group by deptno;  
deptno | string_agg  
-----+-----  
30 | JACK,MAX  
20 | TOM  
(2 rows)
```

Query employees whose work IDs are smaller than 7156:

```
SELECT string_agg(ename,')') FROM employeeinfo where empno < 7156;  
string_agg  
-----  
TOM,JACK  
(1 row)
```

listagg(expression [, delimiter]) WITHIN GROUP(ORDER BY order-list)

Description: Aggregation column data sorted according to the mode specified by **WITHIN GROUP**, and concatenated to a string using the specified delimiter

- **expression**: Mandatory. It specifies an aggregation column name or a column-based, valid expression. It does not support the **DISTINCT** keyword and the **VARIADIC** parameter.
- **delimiter**: Optional. It specifies a delimiter, which can be a string constant or a deterministic expression based on a group of columns. The default value is empty.
- **order-list**: Mandatory. It specifies the sorting mode in a group.

Return type: text

 NOTE

listagg is a column-to-row aggregation function, compatible with Oracle Database 11g Release 2. You can specify the **OVER** clause as a window function. When **listagg** is used as a window function, the **OVER** clause does not support the window sorting or framework of **ORDER BY**, so as to avoid ambiguity in **listagg** and **ORDER BY** of the **WITHIN GROUP** clause.

Example:

The aggregation column is of the text character set type:

```
postgres=#SELECT deptno, listagg(ename, ',') WITHIN GROUP(ORDER BY ename) AS employees FROM
emp GROUP BY deptno;
 deptno |          employees
-----+-----
      10 | CLARK,KING,MILLER
      20 | ADAMS,FORD,JONES,SCOTT,SMITH
      30 | ALLEN,BLAKE,JAMES,MARTIN,TURNER,WARD
(3 rows)
```

The aggregation column is of the integer type:

```
postgres=#SELECT deptno, listagg(mgrno, ',') WITHIN GROUP(ORDER BY mgrno NULLS FIRST) AS mgrnos
FROM emp GROUP BY deptno;
 deptno |          mgrnos
-----+-----
      10 | 7782,7839
      20 | 7566,7566,7788,7839,7902
      30 | 7698,7698,7698,7698,7698,7839
(3 rows)
```

The aggregation column is of the floating point type:

```
postgres=#SELECT job, listagg(bonus, '($); ') WITHIN GROUP(ORDER BY bonus DESC) || '($)' AS bonus
FROM emp GROUP BY job;
 job |          bonus
-----+-----
CLERK | 10234.21($); 2000.80($); 1100.00($); 1000.22($)
PRESIDENT | 23011.88($)
ANALYST | 2002.12($); 1001.01($)
MANAGER | 10000.01($); 2399.50($); 999.10($)
SALESMAN | 1000.01($); 899.00($); 99.99($); 9.00($)
(5 rows)
```

The aggregation column is of the time type:

```
postgres=#SELECT deptno, listagg(hiredate, ', ') WITHIN GROUP(ORDER BY hiredate DESC) AS hiredates
FROM emp GROUP BY deptno;
 deptno |          hiredates
-----+-----
      10 | 1982-01-23 00:00:00, 1981-11-17 00:00:00, 1981-06-09 00:00:00
      20 | 2001-04-02 00:00:00, 1999-12-17 00:00:00, 1987-05-23 00:00:00, 1987-04-19 00:00:00, 1981-12-03
00:00:00
      30 | 2015-02-20 00:00:00, 2010-02-22 00:00:00, 1997-09-28 00:00:00, 1981-12-03 00:00:00, 1981-09-08
00:00:00, 1981-05-01 00:00:00
(3 rows)
```

The aggregation column is of the time interval type.

```
postgres=#SELECT deptno, listagg(vacationTime, ', ') WITHIN GROUP(ORDER BY vacationTime DESC) AS
vacationTime FROM emp GROUP BY deptno;
 deptno |          vacationtime
-----+-----
      10 | 1 year 30 days; 40 days; 10 days
      20 | 70 days; 36 days; 9 days; 5 days
```

```
30 | 1 year 1 mon; 2 mons 10 days; 30 days; 12 days 12:00:00; 4 days 06:00:00; 24:00:00
(3 rows)
```

By default, the delimiter is empty:

```
postgres=#SELECT deptno, listagg(job) WITHIN GROUP(ORDER BY job) AS jobs FROM emp GROUP BY
deptno;
deptno | jobs
-----+-----
10 | CLERKMANAGERPRESIDENT
20 | ANALYSTANALYSTCLERKCLERKMANAGER
30 | CLERKMANAGERSALESMANSALESMANSALESMANSALESMAN
(3 rows)
```

When **listagg** is used as a window function, the **OVER** clause does not support the window sorting of **ORDER BY**, and the **listagg** column is an ordered aggregation of the corresponding groups.

```
postgres=#SELECT deptno, mgrno, bonus, listagg(ename, ';' ) WITHIN GROUP(ORDER BY hiredate)
OVER(PARTITION BY deptno) AS employees FROM emp;
deptno | mgrno | bonus | employees
-----+-----+-----+-----
10 | 7839 | 10000.01 | CLARK; KING; MILLER
10 | 23011.88 | CLARK; KING; MILLER
10 | 7782 | 10234.21 | CLARK; KING; MILLER
20 | 7566 | 2002.12 | FORD; SCOTT; ADAMS; SMITH; JONES
20 | 7566 | 1001.01 | FORD; SCOTT; ADAMS; SMITH; JONES
20 | 7788 | 1100.00 | FORD; SCOTT; ADAMS; SMITH; JONES
20 | 7902 | 2000.80 | FORD; SCOTT; ADAMS; SMITH; JONES
20 | 7839 | 999.10 | FORD; SCOTT; ADAMS; SMITH; JONES
30 | 7839 | 2399.50 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
30 | 7698 | 9.00 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
30 | 7698 | 1000.22 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
30 | 7698 | 99.99 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
30 | 7698 | 1000.01 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
30 | 7698 | 899.00 | BLAKE; TURNER; JAMES; MARTIN; WARD; ALLEN
(14 rows)
```

group_concat(expression [ORDER BY {col_name | expr} [ASC | DESC]] [SEPARATOR str_val])

Description: concatenates the specified **str_val** delimiters used by column data into a string. The concatenation uses a sorting method that must be specified by the **ORDER BY** clause. **ORDER BY 1** is not allowed.

- **expression**: (mandatory) specifies a column name or a column-based valid expression. It does not support the **DISTINCT** keyword or the **VARIADIC** parameter.
- **str_val**: (optional) specifies a delimiter, which can be a string constant or a deterministic expression based on grouped columns. The default value indicates that commas (,) are used as delimiters.

Return type: text

NOTE

The group_concat function is supported only in 8.1.2 or later.

Example:

The default delimiter is a comma (,).

```
postgres=#SELECT group_concat(sname) FROM group_concat_test;
group_concat
```

```
-----  
ADAMS,FORD,JONES,KING,MILLER,SCOTT,SMITH  
(1 row)
```

Delimiters can be customized for the **group_concat** function.

```
postgres=#SELECT group_concat(sname separator ';') from group_concat_test;  
group_concat  
-----  
ADAMS;FORD;JONES;KING;MILLER;SCOTT;SMITH  
(1 row)
```

The **group_concat** function supports the **ORDER BY** clause, which concatenates column data in sequence.

```
postgres=#SELECT group_concat(sname order by snumber separator ';') FROM group_concat_test;  
group_concat  
-----  
MILLER;FORD;SCOTT;SMITH;KING;JONES;ADAMS  
(1 row)
```

covar_pop(Y, X)

Description: Overall covariance

Return type: double precision

Example:

```
postgres=# SELECT COVAR_POP(sr_fee, sr_net_loss) FROM tpceds.store_returns WHERE sr_customer_sk <  
1000;  
covar_pop  
-----  
829.749627587403  
(1 row)
```

covar_samp(Y, X)

Description: Sample covariance

Return type: double precision

Example:

```
postgres=#SELECT COVAR_SAMP(sr_fee, sr_net_loss) FROM tpceds.store_returns WHERE sr_customer_sk <  
1000;  
covar_samp  
-----  
830.052235037289  
(1 row)
```

stddev_pop(expression)

Description: Overall standard difference

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
postgres=#SELECT STDDEV_POP(inv_quantity_on_hand) FROM tpceds.inventory WHERE inv_warehouse_sk =  
1;  
stddev_pop  
-----  
289.224294957556  
(1 row)
```

stddev_samp(expression)

Description: Sample standard deviation of the input values

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
postgres=#SELECT STDDEV_SAMP(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk  
= 1;  
stddev_samp  
-----  
289.224359757315  
(1 row)
```

var_pop(expression)

Description: Population variance of the input values (square of the population standard deviation)

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
postgres=#SELECT VAR_POP(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk = 1;  
var_pop  
-----  
83650.692793695475  
(1 row)
```

var_samp(expression)

Description: Sample variance of the input values (square of the sample standard deviation)

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
postgres=#SELECT VAR_SAMP(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk = 1;  
var_samp  
-----  
83650.730277028768  
(1 row)
```

bit_and(expression)

Description: The bitwise AND of all non-null input values, or null if none

Return type: same as the argument type

Example:

```
postgres=#SELECT BIT_AND(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk = 1;  
bit_and  
-----  
0  
(1 row)
```

bit_or(expression)

Description: The bitwise OR of all non-null input values, or null if none

Return type: same as the argument type

Example:

```
postgres=#SELECT BIT_OR(inv_quantity_on_hand) FROM tpcds.inventory WHERE inv_warehouse_sk = 1;
bit_or
-----
1023
(1 row)
```

bool_and(expression)

Description: Its value is **true** if all input values are **true**, otherwise **false**.

Return type: Boolean

Example:

```
postgres=#SELECT bool_and(100 <2500);
bool_and
-----
t
(1 row)
```

bool_or(expression)

Description: Its value is **true** if at least one input value is **true**, otherwise **false**.

Return type: Boolean

Example:

```
postgres=#SELECT bool_or(100 <2500);
bool_or
-----
t
(1 row)
```

corr(Y, X)

Description: Correlation coefficient

Return type: double precision

Example:

```
postgres=#SELECT CORR(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk < 1000;
corr
-----
0.0381383624904186
(1 row)
```

every(expression)

Description: Equivalent to **bool_and**

Return type: Boolean

Example:

```
postgres=#SELECT every(100 <2500);
every
-----
```

```
t
(1 row)
```

rank(expression)

Description: The tuples in different groups are sorted non-consecutively by **expression**.

Return type: bigint

Example:

```
postgres=#SELECT d_moy, d_fy_week_seq, rank() OVER(PARTITION BY d_moy ORDER BY d_fy_week_seq)
FROM tpceds.date_dim WHERE d_moy < 4 AND d_fy_week_seq < 7 ORDER BY 1,2;
 d_moy | d_fy_week_seq | rank
```

| d_moy | d_fy_week_seq | rank |
|-------|---------------|------|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 2 | 8 |
| 1 | 2 | 8 |
| 1 | 2 | 8 |
| 1 | 2 | 8 |
| 1 | 2 | 8 |
| 1 | 2 | 8 |
| 1 | 2 | 8 |
| 1 | 3 | 15 |
| 1 | 3 | 15 |
| 1 | 3 | 15 |
| 1 | 3 | 15 |
| 1 | 3 | 15 |
| 1 | 3 | 15 |
| 1 | 3 | 15 |
| 1 | 4 | 22 |
| 1 | 4 | 22 |
| 1 | 4 | 22 |
| 1 | 4 | 22 |
| 1 | 4 | 22 |
| 1 | 4 | 22 |
| 1 | 4 | 22 |
| 1 | 5 | 29 |
| 1 | 5 | 29 |
| 2 | 5 | 1 |
| 2 | 5 | 1 |
| 2 | 5 | 1 |
| 2 | 5 | 1 |
| 2 | 5 | 1 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |
| 2 | 6 | 6 |

regr_avgx(Y, X)

Description: Average of the independent variable (**sum(X)/N**)

Return type: double precision

Example:


```
postgres=#SELECT REGR_AVGX(sr_fee, sr_net_loss) FROM tpods.store_returns WHERE sr_customer_sk < 1000;
      regr_avgx
-----
578.606576740795
(1 row)
```

regr_avgy(Y, X)

Description: Average of the dependent variable ($\text{sum}(Y)/N$)

Return type: double precision

Example:

```
postgres=#SELECT REGR_AVGY(sr_fee, sr_net_loss) FROM tpods.store_returns WHERE sr_customer_sk < 1000;
      regr_avgy
-----
50.0136711629602
(1 row)
```

regr_count(Y, X)

Description: Number of input rows in which both expressions are non-null

Return type: bigint

Example:

```
postgres=#SELECT REGR_COUNT(sr_fee, sr_net_loss) FROM tpods.store_returns WHERE sr_customer_sk < 1000;
      regr_count
-----
2743
(1 row)
```

regr_intercept(Y, X)

Description: y-intercept of the least-squares-fit linear equation determined by the (X, Y) pairs

Return type: double precision

Example:

```
postgres=#SELECT REGR_INTERCEPT(sr_fee, sr_net_loss) FROM tpods.store_returns WHERE sr_customer_sk < 1000;
      regr_intercept
-----
49.2040847848607
(1 row)
```

regr_r2(Y, X)

Description: Square of the correlation coefficient

Return type: double precision

Example:

```
postgres=#SELECT REGR_R2(sr_fee, sr_net_loss) FROM tpods.store_returns WHERE sr_customer_sk < 1000;
      regr_r2
-----
```

```
0.00145453469345058
(1 row)
```

regr_slope(Y, X)

Description: Slope of the least-squares-fit linear equation determined by the (X, Y) pairs

Return type: double precision

Example:

```
postgres=#SELECT REGR_SLOPE(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk <
1000;
regr_slope
-----
0.00139920009665259
(1 row)
```

regr_sxx(Y, X)

Description: $\text{sum}(X^2) - \text{sum}(X)^2/N$ (sum of squares of the independent variables)

Return type: double precision

Example:

```
postgres=#SELECT REGR_SXX(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk < 1000;
regr_sxx
-----
1626645991.46135
(1 row)
```

regr_sxy(Y, X)

Description: $\text{sum}(X*Y) - \text{sum}(X) * \text{sum}(Y)/N$ ("sum of products" of independent times dependent variable)

Return type: double precision

Example:

```
postgres=#SELECT REGR_SXY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk < 1000;
regr_sxy
-----
2276003.22847225
(1 row)
```

regr_syy(Y, X)

Description: $\text{sum}(Y^2) - \text{sum}(Y)^2/N$ ("sum of squares" of the dependent variable)

Return type: double precision

Example:

```
postgres=#SELECT REGR_SYY(sr_fee, sr_net_loss) FROM tpcds.store_returns WHERE sr_customer_sk < 1000;
regr_syy
-----
2189417.6547314
(1 row)
```

stddev(expression)

Description: Alias of **stddev_samp**

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
postgres=#SELECT STDDEV(inv_quantity_on_hand) FROM tpods.inventory WHERE inv_warehouse_sk = 1;
stddev
-----
289.224359757315
(1 row)
```

variance(expression)

Description: Alias of **var_samp**

Return type: **double precision** for floating-point arguments, otherwise **numeric**

Example:

```
postgres=#SELECT VARIANCE(inv_quantity_on_hand) FROM tpods.inventory WHERE inv_warehouse_sk = 1;
variance
-----
83650.730277028768
(1 row)
```

checksum(expression)

Description: Returns the CHECKSUM value of all input values. This function can be used to check whether the data in the tables is the same before and after the backup, restoration, or migration of the DWS database. Before and after database backup, database restoration, or data migration, you need to manually run SQL commands to obtain the execution results. Compare the obtained execution results to check whether the data in the tables before and after the backup or migration is the same.

NOTE

- For large tables, the CHECKSUM function may take a long time.
- If the CHECKSUM values of two tables are different, it indicates that the contents of the two tables are different. Using the hash function in the CHECKSUM function may incur conflicts. There is low possibility that two tables with different contents may have the same CHECKSUM value. The same problem may occur when CHECKSUM is used for columns.
- If the time type is timestamp, timestamptz, or smalldatetime, ensure that the time zone settings are the same when calculating the CHECKSUM value.
- If the CHECKSUM value of a column is calculated and the column type can be changed to TEXT by default, set *expression* to the column name.
- If the CHECKSUM value of a column is calculated and the column type cannot be changed to TEXT by default, set *expression* to *Column name::TEXT*.
- If the CHECKSUM value of all columns is calculated, set *expression* to *Table name::TEXT*.

The following types of data can be converted into TEXT types by default: char, name, int8, int2, int1, int4, raw, pg_node_tree, float4, float8, bpchar, varchar, nvarchar2, date, timestamp, timestamptz, numeric, and smalldatetime. Other types need to be forcibly converted to TEXT.

Return type: numeric

Example:

The following shows the CHECKSUM value of a column that can be converted to the TEXT type by default:

```
postgres=#SELECT CHECKSUM(inv_quantity_on_hand) FROM tpcds.inventory;
checksum
-----
24417258945265247
(1 row)
```

CHECKSUM value of a column that cannot be converted to the TEXT type by default (Note that the CHECKSUM parameter is *column_name::TEXT*):

```
postgres=#SELECT CHECKSUM(inv_quantity_on_hand::TEXT) FROM tpcds.inventory;
checksum
-----
24417258945265247
(1 row)
```

The following shows the CHECKSUM value of all columns in a table. Note that the CHECKSUM parameter is set to *Table name::TEXT*. The table name is not modified by its schema.

```
postgres=#SELECT CHECKSUM(inventory::TEXT) FROM tpcds.inventory;
checksum
-----
25223696246875800
(1 row)
```

approx_count_distinct(col_name)

Description: Uses the HyperLogLog++ (HLL++) algorithm to estimate the cardinality (number of rows in a column after deduplication). This function is supported only by clusters of version 8.3.0 or later.

Input parameter description: **col_name** indicates the column whose cardinality needs to be estimated.

NOTE

You can adjust the error rate by setting the **approx_count_distinct_precision** parameter.

- The value range is [10,20]. The default value is 17. The theoretical error rate is 3%.
- This parameter indicates the number of buckets in the HyperLogLog++ (HLL++) algorithm. A larger value of this parameter indicates a larger number of buckets and a smaller theoretical error rate.
- Increasing this parameter value will increase the computing time and memory usage, but they will still be much lower than the cost of using a **COUNT DISTINCT** query. You are advised to use this function to replace the **COUNT DISTINCT** statement when the estimation cardinality is large.

Example:

```
CREATE TABLE employeeinfo (empno smallint, ename varchar(20), job varchar(20), hiredate date,deptno smallint) WITH (ORIENTATION = COLUMN);
INSERT INTO employeeinfo VALUES (7155, 'JACK', 'SALESMAN', '2018-12-01', 30);
INSERT INTO employeeinfo VALUES (7003, 'TOM', 'FINANCE', '2016-06-15', 20);
INSERT INTO employeeinfo VALUES (7357, 'MAX', 'SALESMAN', '2020-10-01', 30);

SELECT APPROX_COUNT_DISTINCT(empno) from employeeinfo;
approx_count_distinct
```

```
-----
              3
(1 row)

SELECT COUNT(DISTINCT empno) FROM employeeinfo GROUP BY ename;
count
-----
      1
      1
      1
(3 rows)
```

UNIQ(col_name)

Description: Aggregate function for calculating the number of non-duplicated values. Similar to **COUNT DISTINCT**, this function calculates the number of rows in a column after deduplication and returns a value indicating the rows without duplication. This function is supported only by clusters of version 8.3.0 or later.

Input parameter description: **col_name** indicates the column for which the number of rows after deduplication needs to be calculated. The value can be SMALLINT, INTEGER, BIGINT, REAL, DOUBLE PRECISION, TEXT, VARCHAR, TIMESTAMP, TIMESTAMPTZ, DATE, TIMETZ or UUID.

NOTE

- Currently, only DWS column-store tables support the **UNIQ** function.
- When the **UNIQ** function is used, the SQL statement must contain **GROUP BY**. To achieve better performance, the GROUP BY fields must be evenly distributed.
- To deduplicate three or more columns in an SQL statement, you can use the **UNIQ** function, which is more efficient than the **COUNT DISTINCT** function.
- Generally, the memory usage of the **UNIQ** function is lower than that of the **COUNT DISTINCT** function. If the memory usage exceeds the threshold when the **COUNT DISTINCT** function is used, you can use the **UNIQ** function to reduce the memory usage.

Example:

```
CREATE TABLE employeeinfo (empno smallint, ename varchar(20), job varchar(20), hiredate date,deptno
smallint) WITH (ORIENTATION = COLUMN);
INSERT INTO employeeinfo VALUES (7155, 'JACK', 'SALESMAN', '2018-12-01', 30);
INSERT INTO employeeinfo VALUES (7003, 'TOM', 'FINANCE', '2016-06-15', 20);
INSERT INTO employeeinfo VALUES (7357, 'MAX', 'SALESMAN', '2020-10-01', 30);

SELECT UNIQ(deptno) FROM employeeinfo GROUP BY ename;
uniq
-----
      1
      1
      1
(3 rows)
```

21.7.12 Window Functions

Regular aggregate functions return a single value calculated from values in a row, or group all rows into a single output row. Window functions perform a calculation across a set of rows and return a value for each row.

A window function call represents the application of an aggregate-like function over some portion of the rows selected by a query. Therefore, aggregate functions ([Aggregate Functions](#)) can also be used as window functions. A window function

can scan all rows and display the raw data and aggregation analysis results at the same time.

Precautions

- Column-store tables support only the window functions **rank (expression)** and **row_number (expression)** and the aggregate functions **sum**, **count**, **avg**, **min**, and **max**. Row-store tables do not have such restrictions.
- A single query can contain one or more window function expressions.
- Window functions can appear only in output columns. If you want to use the values of a window function for condition filtering, you need to nest the window function in the subquery and use the aliases of the window function expression at the outer layer for condition filtering. Example:

```
SELECT classid, id, score FROM (SELECT *, avg(score) OVER(PARTITION BY classid) as avg_score FROM score) WHERE score >= avg_score;
```
- In the query block where the window function is located, the **GROUP BY** expression can be used for grouping and deduplication. In this case, the **PARTITION BY** clause in the window function must be a subset of the **GROUP BY** expression to ensure that the window function performs calculation on the deduplication result. The expression of the **ORDER BY** clause must be a subset of the **GROUP BY** expression or an aggregate function of an aggregate operation. Example:

```
SELECT classid, rank() OVER(PARTITION BY classid ORDER BY sum(score)) as avg_score FROM score GROUP BY classid, id;
```

Syntax

A window function uses the **OVER** clause to define a window. The **OVER** clause is used for grouping data and sorting the elements in a group. Window functions are used for generating sequence numbers for the values in the group.

```
function_name ([expression [, expression ...]]) OVER ( window_definition )  
function_name ([expression [, expression ...]]) OVER window_name  
function_name ( * ) OVER ( window_definition )  
function_name ( * ) OVER window_name
```

window_definition is defined as follows:

```
[ existing_window_name ]  
[ PARTITION BY expression [, ...] ]  
[ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]  
[ frame_clause ]
```

The **PARTITION BY** option specifies that rows with the same **PARTITION BY** expression value are grouped.

The **ORDER BY** option is used to control the order in which the window function processes rows. **ORDER BY** must be followed by a column name. If **ORDER BY** is followed by a number, the number is processed as a constant and does not sort the target column.

frame_clause is defined as follows:

```
[ RANGE | ROWS ] frame_start  
[ RANGE | ROWS ] BETWEEN frame_start AND frame_end
```

When you need to specify a window to calculate the results of all rows in a group, you need to specify the start row and end row of the window range. The window range supports the **RANGE** and **ROWS** modes. The **ROWS** mode specifies the

window by the physical unit (row), and the RANGE mode specifies the window as the logical offset.

In **RANGE** and **ROWS**, you can use **BETWEEN** *frame_start* **AND** *frame_end* to specify the window's first and last rows. If only **frame_start** is specified, **frame_end** is **CURRENT ROW** by default.

The values of **frame_start** and **frame_end** are as follows:

- **CURRENT ROW**: The current row is used as the window frame's start or end point.
- **N PRECEDING**: The window frame starts from the *n*th row to the current row.
- **UNBOUNDED PRECEDING**: The window frame starts at the first row of the partition.
- **N FOLLOWING**: The window frame starts from the current row to the *n*th row.
- **UNBOUNDED FOLLOWING**: The window frame ends with the last row of the partition.

frame_start cannot be **UNBOUNDED FOLLOWING**, *frame_end* cannot be **UNBOUNDED PRECEDING**, and *frame_end* cannot be earlier than *frame_start*. For example, **RANGE BETWEEN CURRENT ROW AND N PRECEDING** is not allowed.

RANK()

Description: The **RANK** function is used for generating non-consecutive sequence numbers for the values in each group. The same values have the same rank value but with sequence numbers.

Return type: bigint

Example:

In the **score(id, classid, score)** table, the rows are student ID, class ID, and exam score.

Use the **RANK** function to sort student scores.

```
CREATE TABLE score(id int,classid int,score int);
INSERT INTO score VALUES(1,1,95),(2,2,95),(3,2,85),(4,1,70),(5,2,88),(6,1,70);
```

```
SELECT id, classid, score,RANK() OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | rank |
|----|---------|-------|------|
| 1 | 1 | 95 | 1 |
| 2 | 2 | 95 | 1 |
| 6 | 1 | 70 | 5 |
| 4 | 1 | 70 | 5 |
| 5 | 2 | 88 | 3 |
| 3 | 2 | 85 | 4 |

(6 rows)

ROW_NUMBER()

Description: The **ROW_NUMBER** function is used for generating consecutive sequence numbers for the values in each group. The same values have different sequence numbers.

Return type: bigint

Example:

```
SELECT id, classid, score, ROW_NUMBER() OVER(ORDER BY score DESC) FROM score ORDER BY score DESC;
```

| id | classid | score | row_number |
|----|---------|-------|------------|
| 1 | 1 | 95 | 1 |
| 2 | 2 | 95 | 2 |
| 5 | 2 | 88 | 3 |
| 3 | 2 | 85 | 4 |
| 6 | 1 | 70 | 5 |
| 4 | 1 | 70 | 6 |

(6 rows)

DENSE_RANK()

Description: The **DENSE_RANK** function is used for generating consecutive sequence numbers for the values in each group. The same values have the same rank value number and the same sequence number.

Return type: bigint

Example:

```
SELECT id, classid, score, DENSE_RANK() OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | dense_rank |
|----|---------|-------|------------|
| 1 | 1 | 95 | 1 |
| 2 | 2 | 95 | 1 |
| 5 | 2 | 88 | 2 |
| 3 | 2 | 85 | 3 |
| 6 | 1 | 70 | 4 |
| 4 | 1 | 70 | 4 |

(6 rows)

PERCENT_RANK()

Description: The **PERCENT_RANK** function is used for generating corresponding sequence numbers for the values in each group. That is, the function calculates the value according to the formula $\text{Sequence number} = (\text{Rank} - 1) / (\text{Total rows} - 1)$. **Rank** is the corresponding sequence number generated based on the **RANK** function for the value and **Total rows** is the total number of elements in a group.

Return type: double precision

Example:

```
SELECT id, classid, score, PERCENT_RANK() OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | percent_rank |
|----|---------|-------|--------------|
| 1 | 1 | 95 | 0 |
| 2 | 2 | 95 | 0 |
| 3 | 2 | 85 | .6 |
| 4 | 1 | 70 | .8 |
| 5 | 2 | 88 | .4 |
| 6 | 1 | 70 | .8 |

(6 rows)

CUME_DIST()

Description: The **CUME_DIST** function is used for generating accumulative distribution sequence numbers for the values in each group. That is, the function calculates the value according to the following formula: $\text{Sequence number} = \text{Number of rows preceding or peer with current row} / \text{Total rows}$.

Return type: double precision

Example:

```
SELECT id,classid,score,CUME_DIST() OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | cume_dist |
|----|---------|-------|-------------------|
| 1 | 1 | 95 | .3333333333333333 |
| 2 | 2 | 95 | .3333333333333333 |
| 5 | 2 | 88 | .5 |
| 3 | 2 | 85 | .6666666666666667 |
| 4 | 1 | 70 | 1 |
| 6 | 1 | 70 | 1 |

(6 rows)

NTILE(num_buckets integer)

Description: The **NTILE** function is used for equally allocating sequential data sets to the buckets whose quantity is specified by **num_buckets** according to **num_buckets integer** and allocating the bucket number to each row. Divide the partition as equally as possible.

Return type: integer

Example:

```
SELECT id,classid,score,NTILE(3) OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | ntile |
|----|---------|-------|-------|
| 1 | 1 | 95 | 1 |
| 2 | 2 | 95 | 1 |
| 5 | 2 | 88 | 2 |
| 3 | 2 | 85 | 2 |
| 4 | 1 | 70 | 3 |
| 6 | 1 | 70 | 3 |

(6 rows)

LAG(value any [, offset integer [, default any]])

Description: The **LAG** function is used for generating lag values for the corresponding values in each group. That is, the value of the row obtained by moving forward the row corresponding to the current value by **offset** (integer) is the sequence number. If the row does not exist after the moving, the result value is the default value. If omitted, **offset** defaults to **1** and **default** to **null**.

Return type: same as the parameter type

Example:

```
SELECT id,classid,score,LAG(id,3) OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | lag |
|----|---------|-------|-----|
| 1 | 1 | 95 | |
| 2 | 2 | 95 | |
| 5 | 2 | 88 | |
| 3 | 2 | 85 | 1 |
| 4 | 1 | 70 | 2 |
| 6 | 1 | 70 | 5 |

(6 rows)

LEAD(value any [, offset integer [, default any]])

Description: The **LEAD** function is used for generating leading values for the corresponding values in each group. That is, the value of the row obtained by

moving backward the row corresponding to the current value by **offset** (integer) is the sequence number. If the number of rows after the moving exceeds the total number for the current group, the result value is the default value. If omitted, **offset** defaults to **1** and **default** to **null**.

Return type: same as the parameter type

Example:

```
SELECT id,classid,score,LEAD(id,3) OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | lead |
|----|---------|-------|------|
| 1 | 1 | 95 | 3 |
| 2 | 2 | 95 | 4 |
| 5 | 2 | 88 | 6 |
| 3 | 2 | 85 | |
| 4 | 1 | 70 | |
| 6 | 1 | 70 | |

(6 rows)

FIRST_VALUE(value any)

Description: The **FIRST_VALUE** function is used for returning the first value of each group.

Return type: same as the parameter type

Example:

```
SELECT id,classid,score,FIRST_VALUE(id) OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | first_value |
|----|---------|-------|-------------|
| 1 | 1 | 95 | 1 |
| 2 | 2 | 95 | 1 |
| 5 | 2 | 88 | 1 |
| 3 | 2 | 85 | 1 |
| 4 | 1 | 70 | 1 |
| 6 | 1 | 70 | 1 |

(6 rows)

LAST_VALUE(value any)

Description: Returns the last value of each group.

Return type: same as the parameter type

The **LAST_VALUE** function supports the IGNORE NULLS syntax. This syntax returns the last value in a non-null window. If all values are **NULL**, **NULL** is returned. The format is as follows:

```
LAST_VALUE (expression) [IGNORE NULLS] OVER (window_definition)
```

Currently, only **ROWS between CURRENT ROW and UNBOUNDED FOLLOWING** and **ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW** are supported.

Example:

```
SELECT id,classid,score,LAST_VALUE(id) OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | last_value |
|----|---------|-------|------------|
| 1 | 1 | 95 | 2 |
| 2 | 2 | 95 | 2 |
| 5 | 2 | 88 | 5 |

| | | | |
|---|---|----|---|
| 3 | 2 | 85 | 3 |
| 4 | 1 | 70 | 6 |
| 6 | 1 | 70 | 6 |

(6 rows)

NTH_VALUE(value any, nth integer)

Description: The *n*th row for a group is the returned value. If the row does not exist, **NULL** is returned by default.

Return type: same as the parameter type

Example:

```
SELECT id,classid,score,NTH_VALUE(id,3) OVER(ORDER BY score DESC) FROM score;
```

| id | classid | score | nth_value |
|----|---------|-------|-----------|
| 1 | 1 | 95 | |
| 2 | 2 | 95 | |
| 5 | 2 | 88 | 5 |
| 3 | 2 | 85 | 5 |
| 4 | 1 | 70 | 5 |
| 6 | 1 | 70 | 5 |

(6 rows)

21.7.13 Type Conversion Functions

cast(x as y)

Description: Converts *x* into the type specified by *y*.

Example:

```
SELECT cast('22-oct-1997' as timestamp);
```

| timestamp |
|---------------------|
| 1997-10-22 00:00:00 |

(1 row)

cast(x, y)

Description: Converts *x* into the type specified by *y*. This function is supported by version 8.2.0 or later clusters.

Example:

```
SELECT cast('22-oct-1997', timestamp);
```

| timestamp |
|---------------------|
| 1997-10-22 00:00:00 |

(1 row)

try_cast(x as type)

Description: Converts *x* to a value of the given type. If the conversion fails and the current type is converted to a value allowed by DWS, **NULL** is returned. Otherwise, an error is raised. This function is supported by version 8.2.0 or later clusters.

Example:

```
SELECT try_cast('a' as int4);
```

| int4 |
|------|
| |

```
-----  
(1 row)  
SELECT try_cast('22-oct-1997' as timestamp);  
       timestamp  
-----  
1997-10-22 00:00:00  
(1 row)
```

hextoraw(string)

Description: Converts a string in hexadecimal format into binary format.

Return type: raw

Example:

```
SELECT hextoraw('7D');  
       hextoraw  
-----  
7D  
(1 row)
```

numtoday(numeric)

Description: Converts values of the number type into the timestamp of the specified type.

Return type: timestamp

Example:

```
SELECT numtoday(2);  
       numtoday  
-----  
2 days  
(1 row)
```

pg_systimestamp()

Description: Obtains the system timestamp.

Return type: timestamp with time zone

Example:

```
SELECT pg_systimestamp();  
       pg_systimestamp  
-----  
2015-10-14 11:21:28.317367+08  
(1 row)
```

rawtohex(string)

Description: Converts a string in binary format into hexadecimal format.

The result represents the ASCII codes of the input characters in hexadecimal format.

Return type: varchar

Example:

```
SELECT rawtohex('1234567');
       rawtohex
-----
31323334353637
(1 row)
```

to_char (datetime/interval [, fmt])

Description: Converts a DATETIME or INTERVAL value of the DATE/TIMESTAMP/TIMESTAMP WITH TIME ZONE/TIMESTAMP WITH LOCAL TIME ZONE type into the VARCHAR type according to the format specified by **fmt**.

- The optional parameter **fmt** includes the following types: date, time, week, quarter, and century. Each type has a unique template. The templates can be combined together. Common templates include: HH, MI, SS, YYYY, MM, and DD. For details, see [Table 21-41](#).
- A template may have a modification word. FM is a common modification word and is used to suppress the preceding zero or the following blank spaces.

Return type: varchar

Example:

```
SELECT to_char(current_timestamp,'HH12:MI:SS');
       to_char
-----
10:19:26
(1 row)
SELECT to_char(current_timestamp,'FMHH12:FMMI:FMSS');
       to_char
-----
10:19:46
(1 row)
```

to_char(double precision, text)

Description: Converts the values of the double-precision type into the strings in the specified format.

Return type: text

Example:

```
SELECT to_char(125.8::real, '999D99');
       to_char
-----
125.80
(1 row)
```

to_char (integer/number[, fmt])

Descriptions: Converts an integer or a value in floating point format into a string in specified format.

- The optional parameter **fmt** can be the following types: decimal characters, grouping characters, positive/negative sign and currency sign. Each type has a unique template. The templates can be combined together. Common templates include: 9, 0, millesimal sign (.), and decimal point (.). For details, see [Table 21-40](#).

- A template can have a modification word, similar to FM. However, FM does not suppress 0 which is output according to the template.
- Use the template X or x to convert an integer value into a string in hexadecimal format.

Return type: varchar

Example:

```
SELECT to_char(1485,'9,999');
to_char
-----
1,485
(1 row)
SELECT to_char( 1148.5,'9,999.999');
to_char
-----
1,148.500
(1 row)
SELECT to_char(148.5,'990999.909');
to_char
-----
0148.500
(1 row)
SELECT to_char(123,'XXX');
to_char
-----
7B
(1 row)
```

to_char(interval, text)

Description: Converts the values of the time interval type into the strings in the specified format.

Return type: text

Example:

```
SELECT to_char(interval '15h 2m 12s', 'HH24:MI:SS');
to_char
-----
15:02:12
(1 row)
```

to_char(int, text)

Description: Converts the values of the integer type into the strings in the specified format.

Return type: text

Example:

```
SELECT to_char(125, '999');
to_char
-----
125
(1 row)
```

to_char(numeric, text)

Description: Converts the values of the numeric type into the strings in the specified format.

Return type: text

Example:

```
SELECT to_char(-125.8, '999D99S');
to_char
-----
125.80-
(1 row)
```

to_char (string)

Description: Converts the CHAR/VARCHAR/VARCHAR2/CLOB type into the VARCHAR type.

If this function is used to convert data of the CLOB type, and the value to be converted exceeds the value range of the target type, an error is returned.

Return type: varchar

Example:

```
SELECT to_char('01110');
to_char
-----
01110
(1 row)
```

to_char(timestamp, text)

Description: Converts the values of the timestamp type into the strings in the specified format.

Return type: text

Example:

```
SELECT to_char(current_timestamp, 'HH12:MI:SS');
to_char
-----
10:55:59
(1 row)
```

to_clob(char/nchar/varchar/nvarchar/varchar2/nvarchar2/text/raw)

Description: Convert the RAW type or text character set type CHAR/NCHAR/VARCHAR/VARCHAR2/NVARCHAR2/TEXT into the CLOB type.

Return type: clob

Example:

```
SELECT to_clob('ABCDEF'::RAW(10));
to_clob
-----
ABCDEF
(1 row)
SELECT to_clob('hello111'::CHAR(15));
to_clob
-----
hello111
(1 row)
SELECT to_clob('gauss123'::NCHAR(10));
to_clob
-----
```

```
-----
gauss123
(1 row)
SELECT to_clob('gauss234'::VARCHAR(10));
to_clob
-----
gauss234
(1 row)
SELECT to_clob('gauss345'::VARCHAR2(10));
to_clob
-----
gauss345
(1 row)
SELECT to_clob('gauss456'::NVARCHAR2(10));
to_clob
-----
gauss456
(1 row)
SELECT to_clob('World222!'::TEXT);
to_clob
-----
World222!
(1 row)
```

to_date(text)

Description: Converts values of the text type into the timestamp in the specified format.

Return type: timestamp

Example:

```
SELECT to_date('2015-08-14');
to_date
-----
2015-08-14 00:00:00
(1 row)
```

to_date(text, text)

Description: Converts the values of the string type into the dates in the specified format.

Return type: timestamp

Example:

```
SELECT to_date('05 Dec 2000', 'DD Mon YYYY');
to_date
-----
2000-12-05 00:00:00
(1 row)
```

to_date(string, fmt)

Description: Converts a string into a value of the DATE type according to the format specified by **fmt**. For details about the fmt format, see [Table 21-41](#).

This function cannot support the CLOB type directly. However, a parameter of the CLOB type can be converted using implicit conversion.

Return type: date

Example:


```
SELECT TO_DATE('05 Dec 2010','DD Mon YYYY');
to_date
-----
2010-12-05 00:00:00
(1 row)
```

to_number (expr [, fmt])

Description: Converts **expr** into a value of the NUMBER type according to the specified format.

For details about the type conversion formats, see [Table 21-40](#).

If a hexadecimal string is converted into a decimal number, the hexadecimal string can include a maximum of 16 bytes if it is to be converted into a sign-free number.

During the conversion from a hexadecimal string to a decimal digit, the format string cannot have a character other than x or X. Otherwise, an error is reported.

Return type: number

Example:

```
SELECT to_number('12,454.8-', '99G999D9S');
to_number
-----
-12454.8
(1 row)
```

to_number(text, text)

Description: Converts the values of the string type into the numbers in the specified format.

Return type: numeric

Example:

```
SELECT to_number('12,454.8-', '99G999D9S');
to_number
-----
-12454.8
(1 row)
```

to_timestamp(double precision)

Description: Converts a UNIX century into a timestamp.

Return type: timestamp with time zone

Example:

```
SELECT to_timestamp(1284352323);
to_timestamp
-----
2010-09-13 12:32:03+08
(1 row)
```

to_timestamp(string [,fmt])

Description: Converts a string into a value of the timestamp type according to the format specified by **fmt**. When **fmt** is not specified, perform the conversion

according to the format specified by **nls_timestamp_format**. For details about the fmt format, see [Table 21-41](#).

In **to_timestamp** in DWS:

- If the input year *YYYY* is 0, an error will be reported.
- If the input year *YYYY* < 0 to specify *SYYYYY* in *fmt*, the year with the value of *n* (an absolute value) BC is output correctly.

Characters in the *fmt* must match the schema for formatting the data and time. Otherwise, an error is reported.

Return type: timestamp without time zone

Example:

```
SHOW nls_timestamp_format;
      nls_timestamp_format
-----
DD-Mon-YYYY HH:MI:SS.FF AM
(1 row)

SELECT to_timestamp('12-sep-2014');
      to_timestamp
-----
2014-09-12 00:00:00
(1 row)
SELECT to_timestamp('12-Sep-10 14:10:10.123000','DD-Mon-YY HH24:MI:SS.FF');
      to_timestamp
-----
2010-09-12 14:10:10.123
(1 row)
SELECT to_timestamp('-1','SYYYYY');
      to_timestamp
-----
0001-01-01 00:00:00 BC
(1 row)
SELECT to_timestamp('98','RR');
      to_timestamp
-----
1998-01-01 00:00:00
(1 row)
SELECT to_timestamp('01','RR');
      to_timestamp
-----
2001-01-01 00:00:00
(1 row)
```

to_timestamp(text, text)

Description: Converts values of the string type into the timestamp of the specified type.

Return type: timestamp

Example:

```
SELECT to_timestamp('05 Dec 2000', 'DD Mon YYYY');
      to_timestamp
-----
2000-12-05 00:00:00
(1 row)
```

The following table describes the value formats of the **to_number** function.

Table 21-40 Template patterns for numeric formatting

| Schema | Description |
|------------|---|
| 9 | Value with specified digits |
| 0 | Values with leading zeros |
| Period (.) | Decimal point |
| Comma (,) | Group (thousand) separator |
| PR | Negative values in angle brackets |
| S | Sign anchored to number (uses locale) |
| L | Currency symbol (uses locale) |
| D | Decimal point (uses locale) |
| G | Group separator (uses locale) |
| MI | Minus sign in the specified position (if the number is less than 0) |
| PL | Plus sign in the specified position (if the number is greater than 0) |
| SG | Plus or minus sign in the specified position |
| RN | Roman numerals (the input values are between 1 and 3999) |
| TH or th | Ordinal number suffix |
| V | Shifts specified number of digits (decimal) |

Table 21-41 describes the patterns of date and time values. They can be used for functions **to_date**, **to_timestamp**, and **to_char**, and the **nls_timestamp_format** parameter.

Table 21-41 Schemas for formatting date and time

| Type | Schema | Description |
|--------|--------|------------------------------------|
| Hour | HH | Number of hours in one day (01-12) |
| | HH12 | Number of hours in one day (01-12) |
| | HH24 | Number of hours in one day (00-23) |
| Minute | MI | Minute (00-59) |
| Second | SS | Second (00-59) |
| | FF | Microsecond (000000-999999) |

| Type | Schema | Description |
|-----------------------|------------|--|
| | SSSSS | Second after midnight (0-86399) |
| Morning and afternoon | AM or A.M. | Morning identifier |
| | PM or P.M. | Afternoon identifier |
| Year | Y,YYY | Year with comma (with four digits or more) |
| | SYYYY | Year with four digits BC |
| | YYYY | Year (with four digits or more) |
| | YYY | Last three digits of a year |
| | YY | Last two digits of a year |
| | Y | Last one digit of a year |
| | IYYY | ISO year (with four digits or more) |
| | IYY | Last three digits of an ISO year |
| | IY | Last two digits of an ISO year |
| | I | Last one digit of an ISO year |
| | RR | <p>Last two digits of a year (A year of the 20th century can be stored in the 21st century.)</p> <p>The password must comply with the following rules:</p> <ul style="list-style-type: none">• If the range of the input two-digit year is between 00 and 49:
If the last two digits of the current year are between 00 and 49, the first two digits of the returned year are the same as the first two digits of the current year.• If the range of the input two-digit year is between 50 and 99:
If the last two digits of the current year are between 00 and 49, the first two digits of the returned year equal to the first two digits of the current year minus 1. <p>If the last two digits of the current year are between 50 and 99, the first two digits of the returned year are the same as the first two digits of the current year.</p> |

| Type | Schema | Description |
|-------------|---|---|
| | RRRR | Capable of receiving a year with four digits or two digits. If there are 2 digits, the value is the same as the returned value of RR. If there are 4 digits, the value is the same as YYYY. |
| | <ul style="list-style-type: none">• BC or B.C.• AD or A.D. | Era indicator Before Christ (BC) and After Christ (AD) |
| Month | MONTH | Full spelling of a month in uppercase (9 characters are filled in if the value is empty.) |
| | MON | Month in abbreviated format in uppercase (with three characters) |
| | MM | Month (01-12) |
| | RM | Month in Roman numerals (I-XII; I=JAN) and uppercase |
| Day | DAY | Full spelling of a date in uppercase (9 characters are filled in if the value is empty.) |
| | DY | Day in abbreviated format in uppercase (with three characters) |
| | DDD | Day in a year (001-366) |
| | DD | Day in a month (01-31) |
| | D | Day in a week (1-7. Sunday is 1.) |
| Week | W | Week in a month (1-5) (The first week starts from the first day of the month.) |
| | WW | Week in a year (1-53) (The first week starts from the first day of the year.) |
| | IW | Week in an ISO year (The first Thursday is in the first week.) |
| Century | CC | Century (with two digits) (The 21st century starts from 2001-01-01.) |
| Julian date | J | Julian date (starting from January 1 of 4712 BC) |
| Quarter | Q | Quarter |

21.7.14 JSON/JSONB Functions and Operators

21.7.14.1 JSON/JSONB Operators

Table 21-42 Common JSON and JSONB Operators

| Operator | Left Operand Type | Right Operand Type | Return Type | Description | Example |
|----------|-------------------|--------------------|-------------|--|--|
| -> | Array-json(b) | int | json(b) | Obtains the array-json element. If the subscript does not exist, NULL is returned. | <pre>SELECT '["a":"foo", {"b":"bar"}, {"c":"baz"}]::json->2; ?column? ----- {"c":"baz"} (1 row)</pre> |
| -> | object-json(b) | text | json(b) | Obtains the value by a key. If no record exists, NULL is returned. | <pre>SELECT '{"a":{"b":"foo"}}::json->'a'; ?column? ----- {"b":"foo"} (1 row)</pre> |
| ->> | Array-json(b) | int | text | Obtains the array-json element. If the subscript does not exist, NULL is returned. | <pre>SELECT '["a":"foo", {"b":"bar"}, {"c":"baz"}]::json->>2; ?column? ----- {"c":"baz"} (1 row)</pre> |
| ->> | object-json(b) | text | text | Obtains the value by a key. If no record exists, NULL is returned. | <pre>SELECT '{"a":{"b":"foo"}}::json->>'a'; ?column? ----- {"b":"foo"} (1 row)</pre> |

| Operator | Left Operand Type | Right Operand Type | Return Type | Description | Example |
|----------|--------------------|--------------------|-------------|---|--|
| #> | container-json (b) | text[] | json | Obtains the JSON object in the specified path. If the path does not exist, NULL is returned.
NOTE
A DWS object identifier can end with a number sign (#). To avoid ambiguity during the parsing of a#>b, you need to add spaces in front of and behind the operator #>. Otherwise, a parsing error is reported. | <pre>SELECT '{"a":{"b":{"c":1}}}'::json #> '{a, b}'; ?column? ----- {"c":1} (1 row)</pre> |
| #>> | container-json (b) | text[] | text | Obtains the JSON object in the specified path. If the path does not exist, NULL is returned. | <pre>SELECT '{"a":{"b":{"c":1}}}'::json #>> '{a, b}'; ?column? ----- {"c":1} (1 row)</pre> |

Table 21-43 Operators supported by jsonb

| Operator | Right Oper and Type | Return Type | Description | Example |
|----------|---------------------|-------------|---|---|
| = | jsonb | bool | Same as the jsonb_eq function, which compares the size of two jsonb files. | <pre>SELECT '{"a":{"b":{"c":1}}}'::jsonb = '{"a":{"b":{"c":1}}}'::jsonb; ?column?</pre> <p>-----</p> <p>t</p> <p>(1 row)</p> |
| <> | jsonb | bool | Same as the jsonb_eq function, which compares the size of two jsonb files. | <pre>SELECT '{"a":{"b":{"c":1}}}'::jsonb <> '{"a":{"b":{"c":1}}}'::jsonb; ?column?</pre> <p>-----</p> <p>f</p> <p>(1 row)</p> |
| < | jsonb | bool | Same as the jsonb_eq function, which compares the size of two jsonb files. | <pre>SELECT '{"a":{"b":{"c":2}}}'::jsonb < '{"a":{"b":{"c":1}}}'::jsonb; ?column?</pre> <p>-----</p> <p>f</p> <p>(1 row)</p> |
| > | jsonb | bool | Same as the jsonb_eq function, which compares the size of two jsonb files. | <pre>SELECT '{"a":{"b":{"c":2}}}'::jsonb > '{"a":{"b":{"c":1}}}'::jsonb; ?column?</pre> <p>-----</p> <p>t</p> <p>(1 row)</p> |
| <= | jsonb | bool | Same as the jsonb_eq function, which compares the size of two jsonb files. | <pre>SELECT '{"a":{"b":{"c":2}}}'::jsonb <= '{"a":{"b":{"c":1}}}'::jsonb; ?column?</pre> <p>-----</p> <p>f</p> <p>(1 row)</p> |
| >= | jsonb | bool | Same as the jsonb_eq function, which compares the size of two jsonb files. | <pre>SELECT '{"a":{"b":{"c":2}}}'::jsonb >= '{"a":{"b":{"c":1}}}'::jsonb; ?column?</pre> <p>-----</p> <p>t</p> <p>(1 row)</p> |
| ? | text | bool | Whether the string of the key or element exists at the top layer of the JSON value. | <pre>SELECT '{"a":1, "b":2}'::jsonb ? 'b'; ?column?</pre> <p>-----</p> <p>t</p> <p>(1 row)</p> |
| ? | text[] | bool | Whether any of these array strings exists as a top-layer key. | <pre>SELECT '{"a":1, "b":2, "c":3, "d":4}'::jsonb ? {'a, b, e'}::text[]; ?column?</pre> <p>-----</p> <p>t</p> <p>(1 row)</p> |

| Operator | Right Oper and Type | Return Type | Description | Example |
|----------|---------------------|-------------|--|--|
| ?& | text[] | bool | Whether all these array strings exist as top-layer keys. | <pre>SELECT '{"a":1, "b":2, "c":3, "d":4}::jsonb ?& {a, b, c}::text[]; ?column? ----- t (1 row)</pre> |
| <@ | jsonb | bool | Whether all items in the JSON file on the left exist at the top layer of the JSON file on the right. | <pre>SELECT '{"b":3}::jsonb <@ '{"a":{"b":{"c":2}}, "b":3}::jsonb; ?column? ----- t (1 row)</pre> |
| @> | jsonb | bool | Whether all items in the JSON file on the right exist at the top layer of the JSON file on the left. | <pre>SELECT '{"a":{"b":{"c":2}}, "b":3}::jsonb @> {"b":3}::jsonb; ?column? ----- t (1 row)</pre> |
| | jsonb | jsonb | Combines two JSONB objects into one. | <pre>SELECT '{"a":1, "b":2}::jsonb '{"c":3, "d":4}::jsonb; ?column? ----- {"a": 1, "b": 2, "c": 3, "d": 4} (1 row)</pre> |
| - | text | jsonb | Deletes a jsonb object and the specified key-value pair. | <pre>SELECT '{"a":1, "b":2}::jsonb - 'a'; ?column? ----- {"b": 2} (1 row)</pre> |
| - | text | jsonb | Deletes a jsonb object and the specified key-value pair. | <pre>SELECT '{"a":1, "b":2, "c":3, "d":4}::jsonb - '{a, b}::text[]; ?column? ----- {"c": 3, "d": 4} (1 row)</pre> |
| - | int | jsonb | Deletes the element corresponding to the subscript in the JSONB array. | <pre>SELECT '['a', "b", "c"]::jsonb - 2; ?column? ----- ["a", "b"] (1 row)</pre> |
| #- | text[] | jsonb | Deletes the key-value pair corresponding to the path in the JSONB object. | <pre>SELECT '{"a":{"b":{"c":{"d":1}}}, "e":2, "f":3}::jsonb #- '{a, b}::text[]; ?column? ----- {"a": {}, "e": 2, "f": 3} (1 row)</pre> |

21.7.14.2 JSON/JSONB Functions

JSON/JSONB functions are used to generate JSON data (see [JSON Types](#)).

NOTE

- Except the **array_to_json** and **row_to_json** functions, all other JSON/JSONB functions and operators are only supported by clusters of version 8.1.2 or later.
- JSON and JSONB provide only basic functions. If there are other complex or special performance requirements, contact Huawei engineers for evaluation.

array_to_json(anyarray [, pretty_bool])

Description: Returns the array as JSON. A multi-dimensional array becomes a JSON array of arrays. Line feeds will be added between dimension-1 elements if **pretty_bool** is **true**.

Return type: json

Example:

```
postgres=#SELECT array_to_json('{{1,5},{99,100}}':int[]);
array_to_json
-----
[[1,5],[99,100]]
(1 row)
```

row_to_json(record [, pretty_bool])

Description: Returns the row as JSON. Line feeds will be added between level-1 elements if **pretty_bool** is **true**.

Return type: json

Example:

```
postgres=#SELECT row_to_json(row(1,'foo'));
row_to_json
-----
{"f1":1,"f2":"foo"}
(1 row)
```

json_agg(any)

Description: Aggregates values into a JSON array.

Return type: array-json

Example:

```
postgres=#SELECT * FROM classes;
name | score
-----+-----
A    | 2
A    | 3
D    | 5
D    |
(4 rows)
postgres=#SELECT name, json_agg(score) score FROM classes group by name order by name;
name | score
-----+-----
A    | [2, 3]
```

```
D | [5, null]
  | [null]
(3 rows)
```

json_object_agg(any, any)

Description: Aggregates values into a JSON object.

Return type: json

Example:

```
postgres=#SELECT * FROM classes;
name | score
-----+-----
A    |    2
A    |    3
D    |    5
D    |
(4 rows)

postgres=#SELECT json_object_agg(name, score) FROM classes group by name order by name;
 json_object_agg
-----
{ "A" : 2, "A" : 3 }
{ "D" : 5, "D" : null }
(2 rows)
```

json_build_array(VARIADIC "any")

Description: Constructs a JSON array that may be of heterogeneous type from a variable parameter list.

Return type: json

Example:

```
postgres=#SELECT json_build_array(1,2,'3',4,5);
 json_build_array
-----
[1, 2, "3", 4, 5]
(1 row)
```

json_build_object(VARIADIC "any")

Description: Constructs a JSON object from a variable parameter list. The parameter list consists of alternate keys and values. The number of input parameters must be an even number. Every two input parameters form a key-value pair. Note that the value of a key cannot be null.

Return type: json

Example:

```
postgres=#SELECT json_build_object('foo',1,'bar',2);
 json_build_object
-----
{"foo" : 1, "bar" : 2}
(1 row)
```

json_object(text[]), json_object(text[], text[])

Description: Constructs a JSON object from a text array.

This is an overloaded function. When the input parameter is a text array, the array length must be an even number, and members are considered alternate key-value pairs. When two text arrays are used, the first array is regarded as a key, and the second array is regarded as a value. The lengths of the two arrays must be the same. Note that the value of a key cannot be null.

Return type: json

Example:

```
postgres=#SELECT json_object('{a, 1, b, "def", c, 3.5}');
          json_object
-----
{"a": "1", "b": "def", "c": "3.5"}
(1 row)

postgres=#SELECT json_object('{{a, 1},{b, "def"}',{c, 3.5}}');
          json_object
-----
{"a": "1", "b": "def", "c": "3.5"}
(1 row)

postgres=#SELECT json_object('{a,b,"a b c"}', '{a,1,1}');
          json_object
-----
{"a": "a", "b": "1", "a b c": "1"}
(1 row)
```

to_json(anyelement)

Description: Converts parameters to json.

Return type: json

Example:

```
postgres=#SELECT to_json('Fred said "Hi."::text);
          to_json
-----
"Fred said \"Hi.\""
(1 row)
-- Convert the column-store table json_tbl_2 to JSON:
postgres=# SELECT * FROM json_tbl_2;
 a | b
---+---
 1 | aaa
 1 | bbb
 2 | ccc
 2 | ddd
(4 rows)
postgres=# SELECT to_json(t.*) FROM json_tbl_2 t;
          to_json
-----
{"a":1,"b":"bbb"}
{"a":2,"b":"ddd"}
{"a":1,"b":"aaa"}
{"a":2,"b":"ccc"}
(4 rows)
```

json_strip_nulls(json)

Description: All object fields with null values are ignored, and other values remain unchanged.

Return type: json

Example:

```
postgres=#SELECT json_strip_nulls('{"f1":1,"f2":null},2,null,3');
json_strip_nulls
-----
{"f1":1},2,null,3
(1 row)
```

json_object_field(json, text)

Description: Same as the operator `->`, which returns the value of a specified key in an object.

Return type: json

Example:

```
postgres=#SELECT json_object_field('{"a": {"b":"foo"}}','a');
json_object_field
-----
{"b":"foo"}
(1 row)
```

json_object_field_text(object-json, text)

Description: Same as the operator `->>`, which returns the value of a specified key in an object.

Return type: text

Example:

```
postgres=#SELECT json_object_field_text('{"a": {"b":"foo"}}','a');
json_object_field_text
-----
{"b":"foo"}
(1 row)
```

json_array_element(array-json, integer)

Description: Same as the operator `->`, which returns the element with the specified subscript in the array.

Return type: json

Example:

```
postgres=#SELECT json_array_element('[1,true,[1,[2,3]],null]',2);
json_array_element
-----
[1,[2,3]]
(1 row)
```

json_array_element_text(array-json, integer)

Description: Same as the operator `->>`, which returns the element with the specified subscript in the array.

Return type: text

Example:

```
postgres=#SELECT json_array_element_text('[1,true,[1,[2,3]],null]',2);
json_array_element_text
-----
[1,[2,3]]
```

```
-----  
[1,[2,3]]  
(1 row)
```

json_extract_path(json, VARIADIC text[])

Description: Same as the operator `#>`, which returns the JSON value of the path specified by `$2`.

Return type: json

Example:

```
postgres=#SELECT json_extract_path('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99,\"f6\":\"stringy\"}}', 'f4','f6');  
json_extract_path  
-----  
\"stringy\"  
(1 row)
```

json_extract_path_text(json, VARIADIC text[])

Description: Same as the operator `#>>`, which returns the text value of the path specified by `$2`.

Return type: text

Example:

```
postgres=#SELECT json_extract_path_text('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99,\"f6\":\"stringy\"}}', 'f4','f6');  
json_extract_path_text  
-----  
stringy  
(1 row)
```

json_array_elements(array-json)

Description: Splits an array. Each element returns a row.

Return type: json

Example:

```
postgres=#SELECT json_array_elements('[1,true,[1,[2,3]],null]');  
json_array_elements  
-----  
1  
true  
[1,[2,3]]  
null  
(4 rows)
```

json_array_elements_text(array-json)

Description: Splits an array. Each element returns a row.

Return type: text

Example:

```
postgres=#SELECT * FROM json_array_elements_text('[1,true,[1,[2,3]],null]');  
value  
-----  
1  
true
```

```
[1,[2,3]]
```

```
(4 rows)
```

json_array_length(array-json)

Description: Returns the array length.

Return type: integer

Example:

```
postgres=#SELECT json_array_length('1,2,3,{"f1":1,"f2":[5,6]},4,null');
json_array_length
```

```
-----
6
```

```
(1 row)
```

json_object_keys(object-json)

Description: Returns all keys at the top layer of the object.

Return type: text

Example:

```
postgres=#SELECT json_object_keys('{"f1":"abc","f2":{"f3":"a","f4":"b"},"f1":"abcd"}');
json_object_keys
```

```
-----
f1
```

```
f2
```

```
f1
```

```
(3 rows)
```

json_each(object-json)

Description: Splits each key-value pair of an object into one row and two columns.

Return type: setof(key text, value json)

Example:

```
postgres=#SELECT * FROM json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
key | value
```

```
-----+-----
f1 | [1,2,3]
```

```
f2 | {"f3":1}
```

```
f4 | null
```

```
(3 rows)
```

json_each_text(object-json)

Description: Splits each key-value pair of an object into one row and two columns.

Return type: setof(key text, value text)

Example:

```
postgres=#SELECT * FROM json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
key | value
```

```
-----+-----
f1 | [1,2,3]
```

```
f2 | {"f3":1}
```

```
f4 |
```

```
(3 rows)
```

json_populate_record(anyelement, object-json [, bool])

Description: *\$1* must be a compound parameter. Each key-value in **object-json** is split. The key is used as the column name to match the column name in *\$1* and fill in the *\$1* format.

NOTE

To define a composite type for the input parameters of a JSON/JSONB function, you can use CREATE TYPE or CREATE TABLE. Here is an example:

```
CREATE TYPE jpop AS (a text, b INT, c timestamp);  
CREATE TABLE jpop2(a text, b INT, c timestamp);
```

Return type: anyelement

Example:

```
postgres=#CREATE TYPE jpop AS (a text, b INT, c timestamp);  
postgres=#SELECT * FROM json_populate_record(null::jpop, '{"a":"blurfl","x":43.2}');  
 a | b | c  
-----+-----+-----  
 blurfl |  |  
(1 row)
```

json_populate_recordset(anyelement, array-json [, bool])

Description: Performs the preceding operations on each element in the *\$2* array by referring to the **json_populate_record** and **jsonb_populate_record** functions. Therefore, each element in the *\$2* array must be of the **object-json** type.

Return type: setof anyelement

Example:

```
postgres=#CREATE TYPE jpop AS (a text, b INT, c timestamp);  
postgres=#SELECT * FROM json_populate_recordset(null::jpop, ' [{"a":1,"b":2}, {"a":3,"b":4}] ');  
 a | b | c  
---+---+---  
 1 | 2 |  
 3 | 4 |  
(2 rows)
```

json_to_record(object-json)

Description: Like all functions that return **record**, the caller must explicitly define the structure of the record using an **AS** clause. The key-value pair of **object-json** is split and reassembled. The key is used as a column name to match and fill in the structure of the record specified by the **AS** clause.

Return type: record

Example:

```
postgres=#SELECT * FROM json_to_record('{"a":1,"b":"foo","c":"bar"}'::json) as x(a int, b text, d text);  
 a | b | d  
---+---+---  
 1 | foo |  
(1 row)
```


json_to_recordset(array-json)

Description: Executes the preceding function on each element in the array by referring to the **json_to_record** function. Therefore, each element in the array must be **object-json**.

Return type: SETOF record

Example:

```
postgres=#SELECT * FROM json_to_recordset('["a":1,"b":{"d":"foo"},"c":true,{"a":2,"c":false,"b":{"d":"bar"}}]') AS x(a INT, b json, c BOOLEAN);
 a |      b      | c
---+-----+---
 1 | {"d":"foo"} | t
 2 | {"d":"bar"} | f
(2 rows)

postgres=#SELECT * FROM json_to_recordset('["a":1,"b":"foo","d":false,{"a":2,"b":"bar","c":true}]') AS x(a INT, b text, c BOOLEAN);
 a | b | c
---+---+---
 1 | foo | 
 2 | bar | t
(2 rows)
```

json_typeof(json)

Description: Checks the JSON type.

Return type: text

Example:

```
postgres=#SELECT value, json_typeof(value) from (values (json '123.4'), (json '"foo"'), (json 'true'), (json 'null'), (json '[1, 2, 3]'), (json '{"x":"foo", "y":123}'), (NULL::json)) as data(value);
 value | json_typeof
-----+-----
 123.4 | number
 "foo" | string
 true  | boolean
 null  | null
 [1, 2, 3] | array
 {"x":"foo", "y":123} | object
(7 rows)
```

jsonb_object(text[])

Description: Constructs an **object-jsonb** from a text array. This is an overloaded function. When the input parameter is a text array, the array length must be an even number, and members are considered alternate key-value pairs.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_object('{a,1,b,2,3,NULL,"d e f","a b c"}');
 jsonb_object
-----
 {"3": null, "a": "1", "b": "2", "d e f": "a b c"}
(1 row)
```

jsonb_object(text[], text[])

Description: When two text arrays are used, the first array is considered a key and the second array is considered a value. The lengths of the two arrays must be the same. Note that the value of a key cannot be null.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_object('{a,b,"a b c"}', '{a,1,1}');
 jsonb_object
-----
{"a": "a", "b": "1", "a b c": "1"}
(1 row)
```

to_jsonb(anyment)

Description: Converts other types to the corresponding jsonb type.

Return type: jsonb

Example:

```
postgres=#SELECT to_jsonb(1.1);
 to_jsonb
-----
1.1
(1 row)
```

jsonb_agg

Description: Aggregates jsonb objects into a jsonb array.

Return type: jsonb

Example:

```
postgres=#SELECT * FROM json_tbl_2;
 a | b
---+---
 1 | aaa
 1 | bbb
 2 | ccc
 2 | ddd
(4 rows)

postgres=#SELECT a, jsonb_agg(b) FROM json_tbl_2 GROUP BY a ORDER BY a;
 a | jsonb_agg
---+-----
 1 | ["aaa", "bbb"]
 2 | ["ccc", "ddd"]
(2 rows)
```

jsonb_object_agg

Description: Aggregates key-value pairs into a JSON object.

Return type: jsonb

Example:

```
postgres=#SELECT * FROM json_tbl_3;
 a | b | c
```

```
-----+-----
1 | aaa | 10
1 | bbb | 20
2 | ccc | 30
2 | ddd | 40
(4 rows)
postgres=#SELECT a, jsonb_object_agg(b, c) FROM json_tbl_3 GROUP BY a ORDER BY a;
a | jsonb_object_agg
-----+-----
1 | {"aaa": 10, "bbb": 20}
2 | {"ccc": 30, "ddd": 40}
(2 rows)
```

jsonb_build_array([VARIADIC "any"])

Description: Constructs a JSON array that may contain heterogeneous types from a variable parameter list.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_build_array('a',1,'b',1.2,'c',true,'d',null,'e',json '{"x": 3, "y": [1,2,3]}',");
jsonb_build_array
-----
["a", 1, "b", 1.2, "c", true, "d", null, "e", {"x": 3, "y": [1, 2, 3]}, null]
(1 row)
```

jsonb_build_object([VARIADIC "any"])

Description: Constructs a JSON object from a variable parameter list. The number of input parameters must be an even number. Every two input parameters form a key-value pair. Note that the value of a key cannot be null.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_build_object(1,2);
jsonb_build_object
-----
{"1": 2}
(1 row)
```

jsonb_strip_nulls(jsonb)

Description: All object fields with null values are omitted. Other null values remain unchanged.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_strip_nulls('["f1":1,"f2":null],2,null,3');
jsonb_strip_nulls
-----
[{"f1": 1}, 2, null, 3]
(1 row)
```

jsonb_object_field(jsonb, text)

Description: Same as the operator ->, which returns the value of a specified key in an object.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_object_field('{"a": {"b":"foo"}}','a');
jsonb_object_field
-----
{"b": "foo"}
(1 row)
```

jsonb_object_field_text(jsonb, text)

Description: Same as the operator ->>, which returns the value of a specified key in an object.

Return type: text

Example:

```
postgres=#SELECT jsonb_object_field_text('{"a": {"b":"foo"}}','a');
jsonb_object_field_text
-----
{"b": "foo"}
(1 row)
```

jsonb_array_element(array-jsonb, integer)

Description: Same as the operator ->, which returns the element with the specified subscript in the array.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_array_element('[1,true,[1,[2,3]],null]',2);
jsonb_array_element
-----
[1, [2, 3]]
(1 row)
```

jsonb_array_element_text(array-jsonb, integer)

Description: Same as the operator ->>, which returns the element with the specified subscript in the array.

Return type: text

Example:

```
postgres=#SELECT jsonb_array_element_text('[1,true,[1,[2,3]],null]',2);
jsonb_array_element_text
-----
[1, [2, 3]]
(1 row)
```

jsonb_extract_path((jsonb, VARIADIC text[]))

Description: Same as the operator #>, which returns the value of the path specified by \$2.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_extract_path('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99,\"f6\":\"stringy\"}}', 'f4','f6');
jsonb_extract_path
-----
\"stringy\"
(1 row)
```

jsonb_extract_path_text((jsonb, VARIADIC text[]))

Description: Same as the operator `#>>`, which returns the value of the path specified by `$2`.

Return type: text

Example:

```
postgres=#SELECT jsonb_extract_path_text('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99,\"f6\":\"stringy\"}}', 'f4','f6');
jsonb_extract_path_text
-----
stringy
(1 row)
```

jsonb_extract((jsonb, VARIADIC text[]))

Description: If you enter any object-jsonb or array-jsonb type, the value of the path specified by `$2` is returned. This function is supported only by clusters of version 9.1.0 or later.

Return type: SETOF jsonb

Example:

```
SELECT jsonb_extract('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99},{\"f6\":\"stringy\"}}', 'f2','f3');
jsonb_extract
-----
1
(1 row)

SELECT jsonb_extract('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99},{\"f5\":\"stringy\"}}', 'f4','f5');
jsonb_extract
-----
99
\"stringy\"
(2 rows)
```

jsonb_extract_text((jsonb, VARIADIC text[]))

Description: If you enter any object-jsonb or array-jsonb type, the value of the path specified by `$2` is returned. This function is supported only by clusters of version 9.1.0 or later.

Return type: SETOF text

Example:

```
SELECT jsonb_extract_text('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99},{\"f6\":\"stringy\"}}', 'f2','f3');
jsonb_extract_text
-----
1
(1 row)

SELECT jsonb_extract_text('{\"f2\":{\"f3\":1},\"f4\":{\"f5\":99},{\"f5\":\"stringy\"}}', 'f4','f5');
jsonb_extract_text
-----
99
```

```
stringy  
(2 rows)
```

jsonb_array_elements(array-jsonb)

Description: Splits an array. Each element returns a row.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_array_elements('[1,true,[1,[2,3]],null]');  
jsonb_array_elements  
-----  
1  
true  
[1, [2, 3]]  
null  
(4 rows)
```

jsonb_array_elements_text(array-jsonb)

Description: Splits an array. Each element returns a row.

Return type: text

Example:

```
postgres=#SELECT * FROM jsonb_array_elements_text('[1,true,[1,[2,3]],null]');  
value  
-----  
1  
true  
[1, [2, 3]]  
(4 rows)
```

jsonb_array_length(array-jsonb)

Description: Returns the array length.

Return type: integer

Example:

```
postgres=#SELECT jsonb_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4,null]');  
jsonb_array_length  
-----  
6  
(1 row)
```

jsonb_object_keys(object-jsonb)

Description: Returns all keys at the top layer of the object.

Return type: SETOF text

Example:

```
postgres=#SELECT jsonb_object_keys('{ "f1": "abc", "f2": { "f3": "a", "f4": "b" }, "f1": "abcd" }');  
jsonb_object_keys  
-----  
f1  
f2  
(2 rows)
```

jsonb_each(object-jsonb)

Description: Splits each key-value pair of an object into one row and two columns.

Return type: setof(key text, value jsonb)

Example:

```
postgres=#SELECT * FROM jsonb_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
key | value
-----+-----
f1  | [1, 2, 3]
f2  | {"f3": 1}
f4  | null
(3 rows)
```

jsonb_each_text(object-jsonb)

Description: Splits each key-value pair of an object into one row and two columns.

Return type: setof(key text, value text)

Example:

```
postgres=#SELECT * FROM jsonb_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null}');
key | value
-----+-----
f1  | [1, 2, 3]
f2  | {"f3": 1}
f4  | 
(3 rows)
```

jsonb_populate_record(anyelement, object-jsonb [, bool])

Description: *\$1* must be a compound parameter. Each key-value in **object-json** is split. The key is used as the column name to match the column name in *\$1* and fill in the *\$1* format.

Return type: anyelement

Example:

```
postgres=#SELECT * FROM jsonb_populate_record(null::jpop, '{"a":"blurfl","x":43.2}');
a | b | c
-----+-----
blurfl | | 
(1 row)
```

jsonb_populate_recordset(anyelement, array-jsonb [, bool])

Description: Performs the preceding operations on each element in the *\$2* array by referring to the **json_populate_record** and **jsonb_populate_record** functions. Therefore, each element in the *\$2* array must be of the **object-json** type.

Return type: setof anyelement

Example:

```
postgres=#SELECT * FROM jsonb_populate_recordset(null::jpop, '[{"a":1,"b":2},{\"a\":3,\"b\":4}]');
a | b | c
-----+-----
1 | 2 | 
3 | 4 | 
(2 rows)
```

jsonb_to_record(object-json)

Description: Like all functions that return **record**, the caller must explicitly define the structure of the record using an **AS** clause. The key-value pair of **object-json** is split and reassembled. The key is used as a column name to match and fill in the structure of the record specified by the **AS** clause.

Return type: record

Example:

```
postgres=#SELECT * FROM jsonb_to_record('{"a":1,"b":"foo","c":"bar"}'::jsonb) as x(a int, b text, d text);
 a | b | d
---+---+---
 1 | foo | 
(1 row)
```

jsonb_to_recordset(array-json)

Description: Executes the preceding function on each element in the array by referring to the **jsonb_to_record** function. Therefore, each element in the array must be **object-jsonb**.

Return type: SETOF record

Example:

```
postgres=#SELECT * FROM jsonb_to_recordset('["a":1,"b":"foo","d":false,{"a":2,"b":"bar","c":true}]') AS x(a
INT, b text, c boolean);
 a | b | c
---+---+---
 1 | foo | 
 2 | bar | t
(2 rows)
```

jsonb_typeof(jsonb)

Description: Checks the JSONB type.

Return type: text

Example:

```
postgres=#SELECT jsonb_typeof(to_jsonb(1.1));
 jsonb_typeof
-----
 number
(1 row)
```

jsonb_ne(jsonb, jsonb)

Description: Same as the operator <>, which compares two values.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_ne('{"a":1,"b":2}'::jsonb, '{"a":1,"b":3}'::jsonb);
 jsonb_ne
-----
 t
(1 row)
```


jsonb_lt(jsonb, jsonb)

Description: Same as the operator <, which compares two values.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_lt('{"a":1, "b":2}::jsonb, '{"a":1, "b":3}::jsonb);
jsonb_lt
-----
t
(1 row)
```

jsonb_gt(jsonb, jsonb)

Description: Same as the operator >, which compares two values.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_gt('{"a":1, "b":2}::jsonb, '{"a":1, "b":3}::jsonb);
jsonb_gt
-----
f
(1 row)
```

jsonb_le(jsonb, jsonb)

Description: Same as the operator <=, which compares two values.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_le('["a", "b"]', '{"a":1, "b":2}');
jsonb_le
-----
t
(1 row)
```

jsonb_ge(jsonb, jsonb)

Description: Same as the operator >=, which compares two values.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_ge('["a", "b"]', '{"a":1, "b":2}');
jsonb_ge
-----
f
(1 row)
```

jsonb_eq(jsonb, jsonb)

Description: Same as the operator =, which compares two values.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_eq('["a", "b"]', '{"a":1, "b":2}');
jsonb_eq
-----
f
(1 row)
```

jsonb_cmp(jsonb, jsonb)

Description: Compares values. A positive value indicates greater than, a negative value indicates less than, and **0** indicates equal.

Return type: integer

Example:

```
postgres=#SELECT jsonb_cmp('["a", "b"]', '{"a":1, "b":2}');
jsonb_cmp
-----
-1
(1 row)
```

jsonb_exists(jsonb, text)

Description: Same as the operator **?**, which determines whether all elements in the string array *\$2* exist at the top layer of *\$1* in the form of **key\elem\scalar**.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_exists('["1",2,3]', '1');
jsonb_exists
-----
t
(1 row)
```

jsonb_exists_any(jsonb, text[])

Description: Same as the operator **?|**, which determines whether all elements in the string array *\$2* exist at the top layer of *\$1* in the form of **key\elem\scalar**.

Return type:

Example:

```
postgres=#SELECT jsonb_exists_any('["1","2",3]', '{1, 2, 4}');
jsonb_exists_any
-----
t
(1 row)
```

jsonb_exists_all(jsonb, text[])

Description: Same as the operator **?&**, which determines whether all elements in the string array *\$2* exist at the top layer of *\$1* in the form of **key\elem\scalar**.

Return type:

bool

Example:

```
postgres=#SELECT jsonb_exists_all('["1","2",3]', '{1, 2}');
jsonb_exists_all
```

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

jsonb_contained(jsonb, jsonb)

Description: Same as the operator <@, which determines whether all elements in the string array \$1 exist at the top layer of \$2.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_contained('[1,2,3]', '[1,2,3,4]');  
jsonb_contained  
-----  
t  
(1 row)
```

jsonb_contains(jsonb, jsonb)

Description: Same as the operator @>, which determines whether all elements in the string array \$2 exist at the top layer of \$1.

Return type: Boolean

Example:

```
postgres=#SELECT jsonb_contains('{"a":1, "b":2, "c":3}::jsonb', '{"a":1}');  
jsonb_contains  
-----  
t  
(1 row)
```

jsonb_concat(jsonb, jsonb)

Description: Combines two JSONB objects into one.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_concat('{"a":1, "b":2}::jsonb', '{"c":3, "d":4}::jsonb');  
jsonb_concat  
-----  
{"a": 1, "b": 2, "c": 3, "d": 4}  
(1 row)
```

jsonb_delete(jsonb, text)

Description: Deletes the key-value pair corresponding to the key value in jsonb.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_delete('{"a":1, "b":2}::jsonb', 'a');  
jsonb_delete  
-----  
{"b": 2}  
(1 row)
```

jsonb_delete_idx(jsonb, text)

Description: Deletes the element corresponding to an array subscript.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_delete_idx('[0,1,2,3,4]::jsonb, 2);
jsonb_delete_idx
-----
[0, 1, 3, 4]
(1 row)
```

jsonb_delete_array(jsonb, VARIADIC text[])

Description: Deletes multiple elements from the jsonb array.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_delete_array('["a", "b", "c"]::jsonb , 'a', 'b');
jsonb_delete_array
-----
["c"]
(1 row)
```

jsonb_delete_path(jsonb, text[])

Description: Deletes elements of a specified path from the jsonb array.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_delete_path('{ "a":{ "b":{ "c":1, "d":2}}, "e":3}::jsonb , array[\'a\', \'b\']);
jsonb_delete_path
-----
{"a": {}, "e": 3}
(1 row)
```

jsonb_set(target jsonb, path text[], new_value jsonb [, create_missing boolean])

Description: Returns *target* with the section designated by *path* replaced by *new_value*, or with *new_value* added if **create_missing** is **true** (**true** by default) and the item designated by *path* does not exist. As with the path-oriented operators, negative integers that appear in *path* count from the end of JSON arrays.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_set('{"f1":1,"f2":null},2,null,3]', '{0,f1}','[2,3,4]', false);
jsonb_set
-----
[{"f1": [2, 3, 4], "f2": null}, 2, null, 3]
(1 row)
```

jsonb_pretty(jsonb)

Description: Returns in indented JSON text.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_pretty('{\"a\":{\"b\":{\"c\":1, \"d\":2}}, \"e\":3}');
jsonb_pretty
-----
{
  \"a\": {
    \"b\": {
      \"c\": 1,
      \"d\": 2
    }
  },
  \"e\": 3
}
(1 row)
```

jsonb_insert(target jsonb, path text[], new_value jsonb [, insert_after boolean])

Description: Returns the target where **new_value** is inserted. When the specified target is located within the **JSONB** array using the path, use the **insert_after** parameter to determine if **new_value** should be inserted after the target. The default value is **false**, meaning that the new value will be inserted after the target. If the target part specified by **path** is in a **JSONB** object, **new_value** is inserted only when the target part does not exist. Like path-oriented operators, a negative integer in **path** indicates that the count starts from the end of the **JSON** array.

Return type: jsonb

Example:

```
postgres=#SELECT jsonb_insert('{\"a\": [0,1,2]}', '{a, 1}', '\"new_value\"');
jsonb_insert
-----
{\"a\": [0, \"new_value\", 1, 2]}
(1 row)
```

ts_headline([config regconfig,] document jsonb, query tsquery [, options text])

Description: Highlights the jsonb search result.

Return type: jsonb

Example:

```
postgres=#SELECT ts_headline('english',
'[{\"id\":\"9928,\"user_id\":\"4562,\"user_name\":\"9LOHR4\",\"create_time\":\"2021-06-22T16:28:16.504518+08:00\"},
{\"id\":\"9959,\"user_id\":\"5524,\"user_name\":\"YID07D\",\"create_time\":\"2021-06-22T16:28:16.557228+08:00\"},
{\"id\":\"9962,\"user_id\":\"7991,\"user_name\":\"7C6QOM\",\"create_time\":\"2021-06-22T16:28:16.56234+08:00\"}]';:jsonb,
to_tsquery('english', '9LOHR4'), 'StartSel = <, StopSel = >');
ts_headline
-----

```

```
[{"id": 9928, "user_id": 4562, "user_name": "<9LOHR4>", "create_time":  
"2021-06-22T16:28:16.504518+08:00"}, {"id": 9959, "user_id": 5524, "user_name": "YID07D", "create_time":  
"2021-06-22T16:28:16.557228+08:00"}, {"id": 9962, "user_id": 7991, "user_name": "7C6QOM",  
"create_time": "2021-06-22T16:28:16.56234+08:00"}]  
(1 row)
```

json_to_tsvector(config regconfig,] json, jsonb)

Description: Converts the json format to the tsvector file format that supports full-text search.

Return type: jsonb

Example:

```
postgres=#SELECT json_to_tsvector('{"a":1, "b":2, "c":3}':json, to_jsonb('key':text));  
json_to_tsvector  
-----  
'b':2 'c':4  
(1 row)
```

21.7.15 Security Functions

gs_password_deadline()

Description: Displays the time before the password of the current account expires. After the password expires, the system prompts the user to change the password. This parameter is related to the GUC parameter **password_effect_time**.

Return type: interval

Example:

```
SELECT gs_password_deadline();  
gs_password_deadline  
-----  
83 days 17:44:32.196094  
(1 row)
```

gs_password_expiration()

Description: Displays the time before the password of the current account expires. After the password expires, the user cannot log in to the database. This parameter is related to the DDL statement **PASSWORD EXPIRATION period** for creating a user. The return value of the function is greater than or equal to -1. If **PASSWORD EXPIRATION period** is not specified during user creation, the default value is -1, indicating that there is no expiration limit.

Return type: interval

Example:

```
SELECT gs_password_expiration();  
gs_password_expiration  
-----  
29 days 23:59:49.731482  
(1 row)
```

login_audit_messages(flag boolean)

Description: Queries login information about a login user.

Return type: tuple

Example:

- Checks the date, time, and IP address successfully authenticated during the last login.

```
SELECT * FROM login_audit_messages(true);
username | database | logintime      | type | result | client_conninfo | session_id
-----+-----+-----+-----+-----+-----+-----
dbadmin  | gaussdb | 2017-06-02 15:28:34+08 | login_success | ok    | gsqr@[local]    | 1663745305.140192538154752.coordinator1
(1 row)
```

- Checks the date, time, and IP address that failed to be authenticated during the last login.

```
SELECT * FROM login_audit_messages(false) ORDER BY logintime desc limit 1;
username | database | logintime      | type | result | client_conninfo | session_id
-----+-----+-----+-----+-----+-----+-----
(0 rows)
```

- Checks the number of failed attempts, date, and time since the previous successful authentication.

```
SELECT * FROM login_audit_messages(false);
username | database | logintime      | type | result | client_conninfo | session_id
-----+-----+-----+-----+-----+-----+-----
(0 rows)
```

login_audit_messages_pid(flag boolean)

Description: Queries login information about a login user. Different from **login_audit_messages**, this function queries login information based on **backendid**. Information about subsequent logins of the same user does not alter the query result of previous logins and cannot be found using this function.

Return type: tuple

Example:

- Checks the date, time, and IP address successfully authenticated during the last login.

```
SELECT * FROM login_audit_messages_pid(true);
username | database | logintime      | type | result | client_conninfo | backendid | session_id
-----+-----+-----+-----+-----+-----+-----+-----
dbadmin  | postgres | 2017-06-02 15:28:34+08 | login_success | ok    | gsqr@[local] | 140311900702464 | 1663745305.140192538154752.coordinator1
(1 row)
```

- Checks the date, time, and IP address that failed to be authenticated during the last login.

```
SELECT * FROM login_audit_messages_pid(false) ORDER BY logintime desc limit 1;
username | database | logintime      | type | result | client_conninfo | backendid | session_id
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)
```

- Checks the number of failed attempts, date, and time since the previous successful authentication.

```
SELECT * FROM login_audit_messages_pid(false);
username | database | logintime      | type | result | client_conninfo | backendid | session_id
-----+-----+-----+-----+-----+-----+-----+-----
(0 rows)
```

pg_query_audit()

Description: Displays audit logs of the CN.

Return type: record

The following table describes return columns.

Table 21-44 Columns returned by the pg_query_audit() function

| Column | Type | Description |
|-----------------|--------------------------|--|
| begintime | timestamp with time zone | Operation start time |
| endtime | timestamp with time zone | Operation end time |
| operation_type | text | Operation type. For details, see Table 21-45 . |
| audit_type | text | Audit type. For details, see Table 21-46 . |
| result | text | Operation result |
| username | text | Name of the user who performs the operation |
| database | text | Database name |
| client_conninfo | text | Client connection information, that is, gsql, JDBC, or ODBC. |
| object_name | text | Object name |
| object_details | text | Columns involved in the statement and their types. |
| command_text | text | Command used to perform the operation |
| detail_info | text | Operation details |
| transaction_xid | text | Transaction ID |
| query_id | text | Query ID |
| node_name | text | Node name |
| session_id | text | ID of a session. |
| local_port | text | Local port |
| remote_port | text | Remote port |
| result_rows | text | Number of records returned after the statement is executed. |

| Column | Type | Description |
|------------|------|--|
| error_code | text | Error code returned when a statement execution error occurs. |

Table 21-45 Operation types

| Operation type | Description |
|----------------|--|
| audit_switch | Indicates that the operations of enabling and disabling the audit log function are audited. |
| login_logout | Indicates that user login and log-out operations are audited. |
| system | Indicates that the system startup, shutdown, and instance switchover operations are audited. |
| sql_parse | Indicates that SQL statement parsing operations are audited. |
| user_lock | Indicates that user locking and unlocking operations are audited. |
| grant_revoke | Indicates that user permission granting and revoking operations are audited. |
| violation | Indicates that user's access violation operations are audited. |
| ddl | Indicates that DDL operations are audited. DDL operations are controlled at a fine granularity based on operation objects. Therefore, audit_system_object is used to specify which objects' DDL operations should be audited. Note that the audit function is active whenever audit_system_object is configured, regardless of the ddl setting. |
| dml | Indicates that the DML operations are audited. |
| select | Indicates that the SELECT operations are audited. |
| internal_event | Indicates that internal incident operations are audited. |
| user_func | <p>Indicates that operations related to user-defined functions, stored procedures, and anonymous blocks are audited.</p> <p>NOTE</p> <p>If a user-defined function or stored procedure contains a FETCH statement, the common_text field records the corresponding CURSOR content when the fetch statement is audited.</p> |

| Operation type | Description |
|-----------------|---|
| special_func | Indicates that special function invoking operations are audited. Special functions include pg_terminate_backend and pg_cancel_backend . |
| copy | Indicates that the COPY operations are audited. |
| set | Indicates that the SET operations are audited. |
| transaction | Indicates that transaction operations are audited. |
| vacuum | Indicates that the VACUUM operations are audited. |
| analyze | Indicates that the ANALYZE operations are audited. |
| cursor | Indicates that cursor operations are audited. |
| anonymous_block | Indicates that the anonymous block operations are audited. |
| explain | Indicates that the EXPLAIN operations are audited. |
| show | Indicates that the SHOW operations are audited. |
| lock_table | Indicates that table lock operations are audited. |
| comment | Indicates that the COMMENT operations are audited. |
| preparestmt | Indicates that the PREPARE , EXECUTE , and DEALLOCATE operations are audited. |
| cluster | Indicates that the CLUSTER operations are audited. |
| constraints | Indicates that the CONSTRAINTS operations are audited. |
| checkpoint | Indicates that the CHECKPOINT operations are audited. |
| barrier | Indicates that the BARRIER operations are audited. |
| cleanconn | Indicates that the CLEAN CONNECTION operations are audited. |
| seclabel | Indicates that security label operations are audited. |
| notify | Indicates that the notification operations are audited. |
| load | Indicates that the loading operations are audited. |
| discard | Indicates that all global temporary tables of the current session are audited. |

Table 21-46 audit_type parameters

| Parameter | Description |
|--|--|
| audit_open/
audit_close | Indicates that the audit type is operations enabling or disabling audit logs. |
| user_login/
user_logout | Indicates that the audit type is operations and users with successful login/logout. |
| system_start/
system_stop/
system_recover/
system_switch | Indicates that the audit type is system startup, shutdown, and instance switchover. |
| sql_wait/sql_parse | Indicates that the audit type is SQL statement parsing. |
| lock_user/unlock_user | Indicates that the audit type is successful user locking and unlocking. |
| grant_role/
revoke_role | Indicates that the audit type is user permission granting and revoking. |
| user_violation | Indicates that the audit type is unauthorized user access operations. |
| ddl_database_object | <p>Indicates that successful DDL operations are audited. DDL operations are controlled at a fine granularity based on operation objects. So, audit_system_object is used to control the objects whose DDL operations are to be audited. (The audit function takes effect as long as audit_system_object is configured, no matter whether ddl is set.)</p> <p>For example, ddl_sequence indicates that the audit type is sequence-related operations.</p> |
| dml_action_insert/
dml_action_delete/
dml_action_update/
dml_action_merge/
dml_action_select | Indicates that the audit type is DML operations such as INSERT , DELETE , UPDATE , and MERGE . |
| internal_event | Indicates that the audit type is internal events. |
| user_func | <p>Indicates that the audit type is user-defined functions, stored procedures, or anonymous block operations.</p> <p>NOTE</p> <p>If a user-defined function or stored procedure contains a FETCH statement, the common_text field records the corresponding CURSOR content when the fetch statement is audited.</p> |
| special_func | Indicates that the audit type is special function invocation. Special functions include pg_terminate_backend and pg_cancel_backend . |
| copy_to/copy_from | Indicates that the audit type is COPY operations. |

| Parameter | Description |
|---|---|
| set_parameter | Indicates that the audit type is SET operations. |
| trans_begin/
trans_commit/
trans_prepare/
trans_rollback_to/
trans_release/
trans_savepoint/
trans_commit_prepare
/
trans_rollback_prepare
/trans_rollback | Indicates that the audit type is transaction-related operations. |
| vacuum/vacuum_full/
vacuum_merge | Indicates that the audit type is VACUUM operations. |
| analyze/analyze_verify | Indicates that the audit type is ANALYZE operations. |
| cursor_declare/
cursor_move/
cursor_fetch/
cursor_close | Indicates that the audit type is cursor-related operations. |
| codeblock_execute | Indicates that the audit type is anonymous blocks. |
| explain | Indicates that the audit type is EXPLAIN operations. |
| show | Indicates that the audit type is SHOW operations. |
| lock_table | Indicates that the audit type is table locking operations. |
| comment | Indicates that the audit type is COMMENT operations. |
| prepare/execute/
deallocate | Indicates that the audit type is PREPARE , EXECUTE , or DEALLOCATE operations. |
| cluster | Indicates that the audit type is CLUSTER operations. |
| constraints | Indicates that the audit type is CONSTRAINTS operations. |
| checkpoint | Indicates that the audit type is CHECKPOINT operations. |
| barrier | Indicates that the audit type is BARRIER operations. |
| cleanconn | Indicates that the audit type is CLEAN CONNECTION operations. |
| seclabel | Indicates that the audit type is security label operations. |
| notify | Indicates that the audit type is notification operations. |
| load | Indicates that the audit type is loading operations. |

| Parameter | Description |
|-----------|---|
| discard | Indicates that the audit type is DISCARD operations. |

pgxc_query_audit()

Description: Displays audit logs of all CNs.

Return type: record

The return fields of this function are the same as those of the **pg_query_audit** function.

pg_query_audit_details()

Description: Views audit logs and parses the **object_name** and **object_details** columns in audit logs from JSON format. This function is supported only by clusters of version 8.2.1.100 or later.

Return type: record

The following table describes return columns.

Table 21-47 Columns returned by the pg_query_audit() function

| Column | Type | Description |
|-----------------|--------------------------|--|
| begintime | timestamp with time zone | Operation start time |
| endtime | timestamp with time zone | Operation end time |
| operation_type | text | Operation type. For details, see Table 21-45 . |
| audit_type | text | Audit type. For details, see Table 21-46 . |
| result | text | Operation result |
| username | text | Name of the user who performs the operation |
| database | text | Database name |
| client_conninfo | text | Client connection information, that is, gsql, JDBC, or ODBC. |
| transaction_xid | text | Transaction ID |
| query_id | text | Query ID |
| node_name | text | Node name |

| Column | Type | Description |
|--------------|------|---|
| session_id | text | ID of a session. |
| local_port | text | Local port |
| remote_port | text | Remote port |
| object_name | text | Table name, function name, and view name. |
| column_name | text | Column name |
| type_of_use | text | Usage type of an object. The options are as follows:
1: reference only (This flag does not exist in actual use.)
2: accessed during execution (columns in the statement and columns accessed during parsing)
4: found in condition statements (excluding condition expressions and functions)
8: found in inner join
16: found in outer join
32: found in aggregation nodes (including distinct , group by , and aggregate functions)
64: found in full join
The values in this column are displayed in overlay mode. |
| use_type | text | Type parsed by type_of_use :
1: Reference only
2: Access
4: Conditional
8: Inner join
16: Outer join
32: Sum
64: Full join |
| command_text | text | Command used to perform the operation |

Example:

1. Query the column names of all objects and object types used in the statement.

```
SET audit_object_details = on;
SELECT object_name,object_details,result_rows,error_code,command_text FROM
pg_query_audit('2023-05-12-03 8:00:00','2023-05-12 22:55:00') where command_text like '%student%';
```

The query result is as follows:

| object_name | object_details | result_rows | error_code | command_text |
|-----------------------|----------------|-------------|------------|--|
| student | | 0 | | CREATE TABLE student(stuNo int, stuName TEXT); |
| studentscore | | 0 | | CREATE TABLE studentscore(stuNo int, stuscore int); |
| public.student_view01 | | 0 | | REPLACE VIEW student_view01 AS SELECT * FROM student t1 where t1.stuNo in (select stuNo from studentscore t2 where t1.stuNo = t2.stuNo); |
| public.student | | 0 | | SELECT * FROM student_view01 |

- The **object_details** column in the query result displays the columns involved in the execution and their usage types. The columns are recorded in JSON format.

You can use the **pg_query_audit_details** function to parse the **object_name** and **object_details** columns.

```
SELECT database,object_name,column_name,type_of_use,use_type FROM
pg_query_audit_details('2021-02-03 8:00:00','2024-02-03 22:55:00','current') where command_text like '%student%';
```

The query result is as follows:

| database | object_name | column_name | type_of_use | use_type |
|-----------|-----------------------|-------------|-------------|--------------------|
| gaussdb | student | | 2 | Access |
| gaussdb | | | 2 | Access |
| gaussdb | studentscore | | 2 | Access |
| gaussdb | | | 2 | Access |
| gaussdb | public.student_view01 | | 2 | Access |
| gaussdb | public.studentscore | | 2 | Access |
| gaussdb | public.student | | 2 | Access |
| gaussdb | | | 2 | Access |
| gaussdb | public.student | stuno | 6 | Access,Conditional |
| gaussdb | public.student | stuname | 2 | Access |
| gaussdb | public.studentscore | stuno | 6 | Access,Conditional |
| gaussdb | public.student_view01 | | 2 | Access |
| gaussdb | public.student | | 2 | Access |
| gaussdb | public.studentscore | | 2 | Access |
| gaussdb | | | 2 | Access |
| (15 rows) | | | | |

pg_delete_audit()

Description: Deletes audit logs generated in a specified period. This function has been deprecated in 8.1.3 and later cluster versions. Return type: void

NOTE

For database security concerns, this function is unavailable. If you call it, the following message is displayed: "ERROR: For security purposes, it is not allowed to manually delete audit logs."

create_audit_csv_foreign_table()

Description: Creates a partitioned foreign table for reading audit logs. You can invoke this function to generate the audit log foreign table **pgxc_audit_logs**. By reading **pgxc_audit_logs**, you can read audit logs stored on OBS. This function is supported only in clusters of 8.3.0 or later.

Input parameter:

- **obs_server**: OBS server name. The value is of the text type.
- **file_path**: OBS file path. The value is of the text type.

Return type: record

Example:

```
-- Create an OBS server.
CREATE SERVER obs_server FOREIGN DATA WRAPPER DFS_FDW OPTIONS (
  address 'obs.example.xxx:xxx',
  encrypt 'true',
  access_key 'xxxxxxxxx',
  secret_access_key 'xxxxxxxxxxxxxxxxx',
  type 'obs' );
-- Invoke this function to generate a foreign table for audit logs.
SELECT * FROM pg_catalog.create_audit_csv_foreign_table('obs_server','/obs-audit/test/');
create_audit_csv_foreign_table
-----
t
(1 row)
-- Read audit logs stored on OBS.
SELECT * FROM pgxc_audit_logs where year=2023 and month=7 and date=24;
 year | month | date |      begintime      |      endtime      | operation_type | audit_type | result
-----+-----+-----+-----+-----+-----+-----+-----+-----
| username | database | client_conninfo |      object_name      | object_details | command_text |
| detail_info | transaction_xid | query_id | node_name | session_id | local_port |
| remote_port | result_rows | error_code |
-----+-----+-----+-----+-----+-----+-----+-----+-----
|-----+-----+-----+-----+-----+-----+-----+-----+-----
|-----+-----+-----+-----+-----+-----+-----+-----+-----
|-----+-----+-----+-----+-----+-----+-----+-----+-----
|-----+-----+-----+-----+-----+-----+-----+-----+-----
|-----+-----+-----+-----+-----+-----+-----+-----+-----
2023 | 7 | 24 | 2023-07-24 09:56:40.727+08 | 2023-07-24 09:56:42.215+08 | dml | |
dml_action_select | ok | dbadmin | gaussdb | gsqr@[local] | public.pgxc_audit_logs | |
select * f
rom pgxc_audit_logs where year=2023 and date=24 and month=7;
```

21.7.16 Conditional Expression Functions

coalesce(expr1, expr2, ..., exprn)

Description: Returns the first argument that is not **NULL** in the argument list.

COALESCE(expr1, expr2) is equivalent to **CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END**.

Examples:

```
SELECT coalesce(NULL,'hello');
 coalesce
-----
hello
(1 row)
```


 NOTE

- **NULL** is returned only if all parameters are **NULL**.
- This value is replaced by the default value when data is displayed.
- Like a CASE expression, COALESCE only evaluates the parameters that are needed to determine the result. That is, parameters to the right of the first non-null parameter are not evaluated.

decode(base_expr, compare1, value1, Compare2,value2, ... default)

Description: Compares base_expr with each compare(n) and returns value(n) if they are matched. If base_expr does not match each **compare(n)**, the default value is returned.

Examples:

```
SELECT decode('A','A',1,'B',2,0);
case
-----
1
(1 row)
```

if(bool_expr, expr1, expr2)

Description: Returns **expr1** or **expr2**. If the value of **bool_expr** is **true**, **expr1** is returned. Otherwise, **expr2** is returned.

This function is equivalent to **CASE WHEN bool_expr = true THEN expr1 ELSE expr2 END**.

Examples:

```
SELECT if(1 < 2, 'yes', 'no');
if
----
yes
(1 row)
```

 NOTE

expr1 and **expr2** can be of any type. For details about the available types, see [UNION, CASE, and Related Constructs](#).

ifnull(expr1, expr2)

Description: Returns **expr1** or **expr2**. If **expr1** is not **NULL**, **expr1** is returned. Otherwise, **expr2** is returned.

This function is logically equivalent to **CASE WHEN expr1 IS NOT NULL THEN expr1 ELSE expr2 END**.

Examples:

```
SELECT ifnull(NULL,'hello');
ifnull
-----
hello
(1 row)
```

 NOTE

expr1 and **expr2** can be of any type. For details about the available types, see [UNION, CASE, and Related Constructs](#).

isnull(expr)

Description: Checks whether **expr** is **NULL**. If it is **NULL**, **true** is returned. Otherwise, **false** is returned.

This function is logically equivalent to **expr IS NULL**.

Examples:

```
SELECT isnull(NULL), isnull('abc');
isnull | isnull
-----+-----
t      | f
(1 row)
```

nullif(expr1, expr2)

Description: Returns **NULL** or **expr1**. If **expr1** is equal to **expr2**, **NULL** is returned. Otherwise, **expr1** is returned.

nullif(expr1, expr2) is equivalent to **CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END**.

Examples:

```
SELECT nullif('hello','world');
nullif
-----
hello
(1 row)
```

Note:

Assume the two parameter data types are different:

- If implicit conversion exists between the two data types, implicitly convert the parameter of lower priority to this data type using the data type of higher priority. If the conversion succeeds, computation is performed. Otherwise, an error is returned. For example:

```
SELECT nullif('1234'::VARCHAR,123::INT4);
nullif
-----
1234
(1 row)
SELECT nullif('1234'::VARCHAR,'2012-12-24'::DATE);
ERROR:  invalid input syntax for type timestamp: "1234"
```

- If implicit conversion is not applied between two data types, an error is displayed. For example:

```
SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE);
ERROR:  operator does not exist: boolean = timestamp without time zone
LINE 1: SELECT nullif(TRUE::BOOLEAN,'2012-12-24'::DATE) FROM DUAL;
      ^
HINT:  No operator matches the given name and argument type(s). You might need to add explicit type casts.
```

nvl(expr1 , expr2)

Description: Returns **expr2** if **expr1** is **NULL**. If **expr1** is not **NULL**, **expr1** is returned.

Examples:

```
SELECT nvl('hello','world');
nvl
-----
hello
(1 row)
```

NOTE

Parameters **expr1** and **expr2** can be of any data type. If **expr1** and **expr2** are of different data types, NVL checks whether **expr2** can be implicitly converted to **expr1**. If it can, the **expr1** data type is returned. If **expr2** cannot be implicitly converted to **expr1** but **expr1** can be implicitly converted to **expr2**, the **expr2** data type is returned. If no implicit type conversion exists between the two parameters and the parameters are different data types, an error is reported.

sys_context('namespace' , 'parameter')

Description: Obtains and returns the parameter values of a specified **namespace**.

Return type: text

Examples:

```
SELECT sys_context('USERENV', 'CURRENT_SCHEMA');
sys_context
-----
public
(1 row)
```

The result varies according to the current actual schema.

NOTE

Currently, only the SYS_CONTEXT('USERENV','CURRENT_SCHEMA') and SYS_CONTEXT('USERENV','CURRENT_USER') formats are supported.

greatest(expr1 [, ...])

Description: Selects the largest value from a list of any number of expressions.

- In Oracle- or Teradata-compatible mode, the returned result is the maximum value of all non-null parameters.
- In MySQL-compatible mode, **null** is returned if an input parameter contains **null**.

Examples:

```
SELECT greatest(1*2,2-3,4-1);
greatest
-----
3
(1 row)
SELECT greatest('ABC', 'BCD', 'CDE');
greatest
-----
```

```
CDE  
(1 row)
```

least(expr1 [, ...])

Description: Selects the smallest value from a list of any number of expressions.

- In Oracle- or Teradata-compatible mode, the returned result is the minimum value of all non-null parameters.
- In MySQL-compatible mode, **null** is returned if an input parameter contains **null**.

Examples:

```
SELECT least(1*2,2-3,4-1);  
least  
-----  
-1  
(1 row)  
SELECT least('ABC','BCD','CDE');  
least  
-----  
ABC  
(1 row)
```

EMPTY_BLOB()

Description: Initiates a BLOB variable in an INSERT or an UPDATE statement to a NULL value.

Return type: BLOB

Examples:

```
-- Create a table:  
CREATE TABLE blob_tb(b blob,id int) DISTRIBUTE BY REPLICATION;  
-- Insert data:  
INSERT INTO blob_tb VALUES (empty_blob(),1);  
--Delete the table.  
DROP TABLE blob_tb;
```

NOTE

The length obtained by using **DBMS.GETLENGTH** is 0.

21.7.17 Range Functions and Operators

21.7.17.1 Range Operators

=

Description: Equals

Example:

```
SELECT int4range(1,5) = '[1,4]::int4range AS RESULT;  
result  
-----  
t  
(1 row)
```

<>

Description: Does not equal to

Example:

```
SELECT numrange(1.1,2.2) <> numrange(1.1,2.3) AS RESULT;
result
-----
t
(1 row)
```

<

Description: Is less than

Example:

```
SELECT int4range(1,10) < int4range(2,3) AS RESULT;
result
-----
t
(1 row)
```

>

Description: Is greater than

Example:

```
SELECT int4range(1,10) > int4range(1,5) AS RESULT;
result
-----
t
(1 row)
```

<=

Description: Is less than or equals

Example:

```
SELECT numrange(1.1,2.2) <= numrange(1.1,2.2) AS RESULT;
result
-----
t
(1 row)
```

>=

Description: Is greater than or equals

Example:

```
SELECT numrange(1.1,2.2) >= numrange(1.1,2.0) AS RESULT;
result
-----
t
(1 row)
```

@>

Description: The object on the left includes the object on the right.

Example:

```
SELECT int4range(2,4) @> int4range(2,3) AS RESULT;
result
-----
t
(1 row)
SELECT '[2011-01-01,2011-03-01)::tsrange @> '2011-01-10)::timestamp AS RESULT;
result
-----
t
(1 row)
```

<@

Description: The object on the right includes the object on the left.

Example:

```
SELECT int4range(2,4) <@ int4range(1,7) AS RESULT;
result
-----
t
(1 row)
SELECT 42 <@ int4range(1,7) AS RESULT;
result
-----
f
(1 row)
```

&&

Description: Overlap (have points in common)

Example:

```
SELECT int8range(3,7) && int8range(4,12) AS RESULT;
result
-----
t
(1 row)
```

<<

Description: Strictly left of

Example:

```
SELECT int8range(1,10) << int8range(100,110) AS RESULT;
result
-----
t
(1 row)
```

>>

Description: Strictly right of

Example:

```
SELECT int8range(50,60) >> int8range(20,30) AS RESULT;
result
-----
t
(1 row)
```

&<

Description: Does not extend to the right of

Example:

```
SELECT int8range(1,20) &< int8range(18,20) AS RESULT;  
result  
-----  
t  
(1 row)
```

&>

Description: Does not extend to the left of

Example:

```
SELECT int8range(7,20) &> int8range(5,10) AS RESULT;  
result  
-----  
t  
(1 row)
```

-|-

Description: Is adjacent to

Example:

```
SELECT numrange(1.1,2.2) -|- numrange(2.2,3.3) AS RESULT;  
result  
-----  
t  
(1 row)
```

+

Description: Union

Example:

```
SELECT numrange(5,15) + numrange(10,20) AS RESULT;  
result  
-----  
[5,20)  
(1 row)
```

Description: Intersection

Example:

```
SELECT int8range(5,15) * int8range(10,20) AS RESULT;  
result  
-----  
[10,15)  
(1 row)
```

-

Description: Difference

Example:

```
SELECT int8range(5,15) - int8range(10,20) AS RESULT;  
result  
-----  
[5,10)  
(1 row)
```

NOTE

- The simple comparison operators <, >, <=, and >= compare the lower bounds first, and only if those are equal, compare the upper bounds.
- The <<, >>, and -|- operators always return false when an empty range is involved; that is, an empty range is not considered to be either before or after any other range.
- The union and difference operators will fail if the resulting range would need to contain two disjoint sub-ranges.

21.7.17.2 Range Functions

lower(anyrange)

Description: Lower bound of range

Return type: Range's element type

Example:

```
SELECT lower(numrange(1.1,2.2)) AS RESULT;  
result  
-----  
1.1  
(1 row)
```

upper(anyrange)

Description: Upper bound of range

Return type: Range's element type

Example:

```
SELECT upper(numrange(1.1,2.2)) AS RESULT;  
result  
-----  
2.2  
(1 row)
```

isempty(anyrange)

Description: Is the range empty?

Return type: boolean

Example:

```
SELECT isempty(numrange(1.1,2.2)) AS RESULT;  
result  
-----  
f  
(1 row)
```


lower_inc(anyrange)

Description: Is the lower bound inclusive?

Return type: boolean

Example:

```
SELECT lower_inc(numrange(1.1,2.2)) AS RESULT;  
result  
-----  
t  
(1 row)
```

upper_inc(anyrange)

Description: Is the upper bound inclusive?

Return type: boolean

Example:

```
SELECT upper_inc(numrange(1.1,2.2)) AS RESULT;  
result  
-----  
f  
(1 row)
```

lower_inf(anyrange)

Description: Is the lower bound infinite?

Return type: boolean

Example:

```
SELECT lower_inf('(,)'::daterange) AS RESULT;  
result  
-----  
t  
(1 row)
```

upper_inf(anyrange)

Description: Is the upper bound infinite?

Return type: boolean

Example:

```
SELECT upper_inf('(,)'::daterange) AS RESULT;  
result  
-----  
t  
(1 row)
```

NOTE

The **lower** and **upper** functions return null if the range is empty or the requested bound is infinite. The **lower_inc**, **upper_inc**, **lower_inf**, and **upper_inf** functions all return false for an empty range.

21.7.18 Data Redaction Functions

Data redaction functions are used to mask and protect sensitive data. Generally, you are advised to bind these functions to the columns to be redacted based on the data redaction syntax, rather than use them directly on query statements.

mask_none(column_name)

Description: Masks no data (for internal tests only).

Return type: same as **column_name**

mask_full(column_name)

Description: Replaces all data with a fixed value. The fixed value varies depending on the data type of the redacted column.

Return type: same as **column_name**

mask_partial(column_name, mask_digital, mask_from[, mask_to])

Description: Replaces the digits from the **mask_from** to **mask_to** position in a number with the digit specified by **mask_digital**. The default value of **mask_to** can be used, which indicates that the digits from the **mask_from** position to the end of the number are replaced. **mask_digital** can only be a digit from 0 to 9.

Return type: same as **column_name**

mask_partial(column_name [, input_format, output_format], mask_char, mask_from[, mask_to])

Description: Replaces the digits from the **mask_from** to **mask_to** position in a string with the character specified by **mask_char** based on the given input and output formats.

Parameter description:

- **input_format**
The input format is a character string of V and F, whose length is the same as that of the data in the redacted column. Characters in positions corresponding to V may be masked, and characters in positions corresponding to F are skipped. The V character string specifies which characters are to be masked. The input and output formats apply to data with a fixed length, such as bank card numbers, ID card numbers, and phone numbers.
- **output_format**
The output format is a character string of V and any other character, whose length is the same as that of the data in the redacted column. V characters correspond to those in the **input_format**, and other characters correspond to the F characters in the **input_format**.
For parameters **input_format** and **output_format**, you can use their default values or set them to "". In this case, there is no requirement for the input or output format, and the whole string will be masked.

- **mask_char**
Masking character, which can be any one character, for example, an asterisk (*) or a number sign (#).
- **mask_from**
First character in the string that will be masked. The value must be greater than 0.
- **mask_to**
Last character in the string that will be masked. The default value can be used, which indicates that the character from the **mask_from** position to the last character of the string will be masked.

Return type: same as **column_name**

mask_partial(column_name, mask_field1, mask_value1, mask_field2, mask_value2, mask_field3, mask_value3)

Description: Masks a date or time based on three specified fields. If **mask_value** is -1, the corresponding **mask_field** is not masked. **mask_field** can be **month**, **day**, **year**, **hour**, **minute**, or **second**. The value range of each field must be within that of the actual time unit.

Return type: same as **column_name**

NOTE

Redaction functions are recommended if you want to create redaction policies.

For how to use data masking functions, see the examples in "Database Security Management" > "Sensitive Data Management" > "Data Masking" in the *Data Warehouse Service (DWS) Developer Guide*.

User-Defined Redaction Functions

You can use the PL/pgSQL language to customize redaction functions.

User-defined redaction functions must meet the following requirements:

- The return type must be the same as the data type of the redacted column.
- The functions can be pushed down.
- In addition to the redaction format, only one column can be specified in the argument list for data redaction.
- The functions only implement the formatting for specific data types and do not involve complex association operations with other table objects.

If either of the first two requirements is not met, an error will be reported when you create a redaction policy. If either of the last two requirements is not met, unexpected problems may occur in query execution results.

Deleting the bound user-defined data redaction function will cause an error during query execution due to an invalid redaction policy.

21.7.19 Bitmap Functions and Operators

21.7.19.1 Roaring Bitmap Operators

Starting from DWS 8.1.3, user profiling and precision marketing scenarios benefit from the utilization of efficient bitmap processing operators. This enhancement significantly enhances query performance.

=

Description: Compares two Roaring bitmaps to check whether they are equal.

Return type: bool

Example:

```
SELECT rb_build('{1,2,3}') = rb_build('{1,2,3}');
?column?
-----
t
(1 row)
SELECT rb_build('{2,3}') = rb_build('{1,2,3}');
?column?
-----
f
(1 row)
```

<>

Description: Compares two Roaring bitmaps to check whether they are unequal.

Return type: bool

Example:

```
SELECT rb_build('{1,2,3}') <> rb_build('{1,2,3}');
?column?
-----
f
(1 row)
SELECT rb_build('{2,3}') <> rb_build('{1,2,3}');
?column?
-----
t
(1 row)
```

&

Description: Calculates the intersection of two Roaring bitmaps.

Return type: RoaringBitmap

Example:

```
SELECT rb_to_array(rb_build('{2,3}') & rb_build('{1,2,3}'));
rb_to_array
-----
{2,3}
(1 row)
```

|

Description: Calculates the union of two Roaring bitmaps.

Return type: RoaringBitmap

Example:

```
SELECT rb_to_array(rb_build('{2,3}') | rb_build('{1,2,3}'));
rb_to_array
-----
{1,2,3}
(1 row)
```

|

Description: Calculates the result of adding an ID to a Roaring bitmap.

Return type: RoaringBitmap

Example:

```
SELECT rb_to_array(rb_build('{2,3}') | 4);
rb_to_array
-----
{2,3,4}
(1 row)
```

#

Description: XOR result of two Roaring bitmaps.

Return type: RoaringBitmap

Example:

```
SELECT rb_to_array(rb_build('{2,3}') # rb_build('{1,2,3}'));
rb_to_array
-----
{1}
(1 row)
```

-

Description: Calculates the result set in the first Roaring bitmap but not in the second Roaring bitmap.

Return type: RoaringBitmap

Example:

```
SELECT rb_to_array(rb_build('{2,3,4}') - rb_build('{1,2,3}'));
rb_to_array
-----
{4}
(1 row)
```

-

Description: The result set of removing a specified ID from a Roaring bitmap.

Return type: RoaringBitmap

Example:

```
SELECT rb_to_array(rb_build('{2,3,4}') - 3);
rb_to_array
-----
{2,4}
(1 row)
```

@>

Description: Determines whether the Roaring bitmap before an operator contains the Roaring bitmap after the operator.

Return type: bool

Example:

```
SELECT rb_build('{2,3,4}') @> rb_build('{2,3}');
?column?
-----
t
(1 row)
SELECT rb_build('{2,3,4}') @> 4;
?column?
-----
t
(1 row)
```

<@

Description: Determines whether the Roaring bitmap before an operator is contained in the Roaring bitmap after the operator.

Return type: bool

Example:

```
SELECT 4 <@ rb_build('{2,3,4}');
?column?
-----
t
(1 row)
SELECT rb_build('{2,3,4}') <@ rb_build('{2,3}');
?column?
-----
f
(1 row)
```

&&

Description: If two Roaring bitmaps overlap, **true** is returned. Otherwise, **false** is returned.

Return type: bool

Example:

```
SELECT rb_build('{2,3,4}') && rb_build('{2,3}');
?column?
-----
t
(1 row)
SELECT rb_build('{2,3,4}') && rb_build('{7,8,9}');
?column?
-----
f
(1 row)
```

21.7.19.2 Bitmap Functions

Since 8.1.3, DWS supports efficient bitmap processing functions and operators, which can be used in user profiling and precision marketing, greatly improving query performance.

rb_build(array)

Description: Converts an int array to the RoaringBitmap type.

Return type: RoaringBitmap

Example:

```
SELECT rb_build('{1,2,3}');
rb_build
-----
\3a300000010000000000020010000000010002000300
(1 row)
CREATE TABLE r_row (a int, b text, c roaringbitmap);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

INSERT INTO r_row values (1, 'a', rb_build('{1,2,3}'));
INSERT 0 1

SELECT * FROM r_row;
 a | b | c
---+---+---
 1 | a | \3a300000010000000000020010000000010002000300
(1 row)

INSERT INTO r_row values (2, 'b', rb_build('{}'));
INSERT 0 1

SELECT * FROM r_row;
 a | b | c
---+---+---
 2 | b | \3a3000000000000000
 1 | a | \3a300000010000000000020010000000010002000300
(2 rows)
```

rb_iterate(roaringbitmap)

Description: Converts roaringbitmap data into int data and outputs the data in multiple lines.

Return type: record (int value in multiple rows)

Example:

```
postgres=#SELECT rb_iterate(c) FROM r_row;
rb_iterate
-----
1
2
3
(3 rows)
```

rb_to_array(roaringbitmap)

Description: Using rb_build reverse operation to convert roaringBitmap into an int array.

Return type: array

Example:

```
postgres=#SELECT rb_to_array(c) FROM r_row;
rb_to_array
-----
{1,2,3}
(1 row)
postgres=#SELECT rb_to_array('\x3a30000001000000000020010000000010002000300');
rb_to_array
-----
{1,2,3}
(1 row)
```

rb_and(roaringbitmap, roaringbitmap)

Description: Calculates the intersection of two Roaring bitmaps.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT rb_to_array(rb_and(rb_build('{1,2,3}'), rb_build('{2,3,4}')));
rb_to_array
-----
{2,3}
(1 row)
```

rb_or(roaringbitmap, roaringbitmap)

Description: Calculates the union of two Roaring bitmaps.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT rb_to_array(rb_or(rb_build('{1,2,3}'), rb_build('{2,3,4}')));
rb_to_array
-----
{1,2,3,4}
(1 row)
```

rb_xor(roaringbitmap, roaringbitmap)

Description: Calculates the XOR of two Roaring bitmaps.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT rb_to_array(rb_xor(rb_build('{1,2,3}'), rb_build('{2,3,4}')));
rb_to_array
-----
{1,4}
(1 row)
```

rb_andnot(roaringbitmap, roaringbitmap)

Description: Sets in the first Roaring bitmap set but not in the second Roaring bitmap set.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT rb_to_array(rb_andnot(rb_build('{1,2,3}'), rb_build('{2,3,4}')));
rb_to_array
-----
{1}
(1 row)
```

rb_cardinality(roaringbitmap)

Description: Calculates the cardinality of a Roaring bitmap.

Return type: int

Example:

```
postgres=#SELECT rb_cardinality(rb_build('{1,2,3}'));
rb_cardinality
-----
3
(1 row)
```

rb_and_cardinality(roaringbitmap, roaringbitmap)

Description: Calculates the cardinality of the intersection of two Roaring bitmaps.

Return type: int

Example:

```
postgres=#SELECT rb_and_cardinality(rb_build('{1,2,3}'), rb_build('{2,3,4}'));
rb_and_cardinality
-----
2
(1 row)
```

rb_or_cardinality(roaringbitmap, roaringbitmap)

Description: Calculates the cardinality of the union of two Roaring bitmaps.

Return type: int

Example:

```
postgres=#SELECT rb_or_cardinality(rb_build('{1,2,3}'), rb_build('{2,3,4}'));
rb_or_cardinality
-----
4
(1 row)
```

rb_xor_cardinality(roaringbitmap, roaringbitmap)

Description: Calculates the cardinality of two Roaring bitmaps after the XOR operation.

Return type: int

Example:

```
postgres=#SELECT rb_xor_cardinality(rb_build('{1,2,3}'), rb_build('{2,3,4}'));
rb_xor_cardinality
-----
2
(1 row)
```

rb_andnot_cardinality(roaringbitmap, roaringbitmap)

Description: Calculates the cardinality of two Roaring bitmaps after ANDNOT operation.

Return type: int

Example:

```
postgres=#SELECT rb_andnot_cardinality(rb_build('{1,2,3}'), rb_build('{2,3,4}'));
rb_andnot_cardinality
-----
1
(1 row)
```

rb_is_empty(roaringbitmap)

Description: Determines whether a Roaring bitmap is empty.

Return type: bool

Example:

```
postgres=#SELECT rb_is_empty(rb_build('{1,2,3}'));
rb_is_empty
-----
f
(1 row)
```

rb_equals(roaringbitmap, roaringbitmap)

Description: Determines whether two Roaring bitmaps are equal.

Return type: bool

Example:

```
postgres=#SELECT rb_equals(rb_build('{1,2,3}'), rb_build('{2,3,4}'));
rb_equals
-----
f
(1 row)
```

rb_intersect(roaringbitmap, roaringbitmap)

Description: Determines whether two Roaring bitmaps are intersected.

Return type: bool

Example:

```
postgres=#SELECT rb_intersect(rb_build('{1,2,3}'), rb_build('{2,3,4}'));
rb_intersect
-----
t
(1 row)
```

rb_min(roaringbitmap)

Description: Returns the minimum value in a Roaring bitmap.

Return type: int

Example:

```
postgres=#SELECT rb_min(rb_build('{1,2,3}'));
rb_min
-----
1
(1 row)
```

rb_max(roaringbitmap)

Description: Returns the maximum value in a Roaring bitmap.

Return type: int

Example:

```
postgres=#SELECT rb_max(rb_build('{1,2,3}'));
rb_max
-----
3
(1 row)
```

rb_add(roaringbitmap, int)

Description: Adds an element to a Roaring bitmap.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT rb_to_array(rb_add(rb_build('{1,3}'), 2));
rb_to_array
-----
{1,2,3}
(1 row)
```

rb_added(int, roaringbitmap)

Description: Adds an element to a Roaring bitmap.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT rb_to_array(rb_added(2, rb_build('{1,3}')));
rb_to_array
-----
{1,2,3}
(1 row)
```

rb_contain(roaringbitmap,int)

Description: Determines whether a Roaring bitmap contains the specified element.

Return type: bool

Example:

```
postgres=#SELECT rb_contain(rb_build('{1,3}'), 2);
rb_contain
-----
f
(1 row)
```

rb_containedby(int,roaringbitmap)

Description: Determines whether the given element is included in a given Roaring bitmap.

Example:

```
postgres=#SELECT rb_containedby(2,rb_build('{1,3}'));
rb_containedby
-----
f
(1 row)
```

rb_contain_rb(roaringbitmap,roaringbitmap)

Description: Determines whether the first Roaring bitmap contains the second Roaring bitmap.

Return type: bool

Example:

```
postgres=#SELECT rb_contain_rb(rb_build('{1,3}'), rb_build('{2,3}'));
rb_contain_rb
-----
f
(1 row)
```

rb_containedby_rb(roaringbitmap,roaringbitmap)

Description: Determines whether the second Roaring bitmap contains the first Roaring bitmap.

Return type: bool

Example:

```
postgres=#SELECT rb_containedby_rb(rb_build('{1,3}'), rb_build('{2,3}'));
rb_containedby_rb
-----
f
(1 row)
```

rb_remove(roaringbitmap,int)

Description: Removes elements from a Roaring bitmap.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT rb_to_array(rb_remove(rb_build('{1,3}'),1));
rb_to_array
-----
{3}
(1 row)
```

rb_clear(roaringbitmap,int,int)

Description: Clears elements within a specified range from roaring bitmaps.

Return type: RoaringBitmap

Example:

```
postgres=#SELECT
rb_to_array(rb_clear(rb_build('{1,2,3}'),1,2));

rb_to_array
-----
{2,3}
(1 row)
```

rb_flip(roaringbitmap,int,int)

Description: Reverses elements in a specified range.

Example:

```
postgres=#SELECT rb_to_array(rb_flip(rb_build('{1,2,3,7,9}'), 1,10));
rb_to_array
-----
{4,5,6,8,10}
(1 row)
```

rb_rank(roaringbitmap,int)

Description: Returns the cardinality of the set of values less than the specified value.

Return type: int

Example:

```
postgres=#SELECT rb_rank(rb_build('{1,10,100}'),99);
rb_rank
-----
2
(1 row)
```

21.7.19.3 Roaring Bitmap Aggregation Functions

Since 8.1.3, DWS supports efficient bitmap processing functions and operators, which can be used in user profiling and precision marketing, greatly improving query performance.

rb_build_agg(int)

Description: Aggregates int values in a group into a RoaringBitmap value.

Return type: RoaringBitmap

Example:

```
CREATE TABLE t1 (a int ,b int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
INSERT INTO t1 SELECT generate_series(1,10),generate_series(1,20,2);
INSERT 0 10
SELECT rb_iterate(rb_build_agg(b)) FROM t1;
rb_iterate
-----
1
```

```
3
5
7
9
11
13
15
17
19
(10 rows)
```

rb_and_agg(roaringbitmap)

Description: Aggregates data of the Roaring bitmaps in a group into a Roaring bitmap set based on the INTERSECT operation.

Example:

```
CREATE TABLE r1(a int ,b roaringbitmap);
INSERT INTO r1 SELECT a, rb_build_agg(b) FROM t1 GROUP BY a;
INSERT INTO t1 SELECT generate_series(1,10),generate_series(1,20,4);
INSERT INTO r1 SELECT a, rb_build_agg(b) FROM t1 GROUP BY a;
SELECT a, rb_to_array(rb_and_agg(b)) FROM r1 GROUP BY a ORDER BY a;
a | rb_to_array
-----+-----
1 | {1}
2 | {3}
3 | {5}
4 | {7}
5 | {9}
6 | {11}
7 | {13}
8 | {15}
9 | {17}
10 | {19}
(10 rows)
```

rb_or_agg(roaringbitmap)

Description: Combines Roaring bitmaps in a group into one Roaring bitmap based on the UNION logic.

Example:

```
SELECT a, rb_to_array(rb_or_agg(b)) FROM r1 GROUP BY a ORDER BY a;
a | rb_to_array
-----+-----
1 | {1}
2 | {3,5}
3 | {5,9}
4 | {7,13}
5 | {9,17}
6 | {1,11}
7 | {5,13}
8 | {9,15}
9 | {13,17}
10 | {17,19}
(10 rows)
```

rb_xor_agg(roaringbitmap)

Description: Combines Roaring bitmaps in a group into one Roaring bitmap based on the XOR logic.

Example:

```
SELECT a, rb_to_array(rb_xor_agg(b)) FROM r1 GROUP BY a ORDER BY a;
```

| a | rb_to_array |
|----|-------------|
| 1 | {} |
| 2 | {5} |
| 3 | {9} |
| 4 | {13} |
| 5 | {17} |
| 6 | {1} |
| 7 | {5} |
| 8 | {9} |
| 9 | {13} |
| 10 | {17} |

(10 rows)

rb_and_cardinality_agg(roaringbitmap)

Description: Cardinality of Roaring bitmaps in a group calculated based on INTERSECT.

Return type: int

Example:

```
SELECT a, rb_and_cardinality_agg(b) FROM r1 GROUP BY a ORDER BY 1;
```

| a | rb_and_cardinality_agg |
|----|------------------------|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 1 |
| 10 | 1 |

(10 rows)

rb_or_cardinality_agg(roaringbitmap)

Description: Cardinality of Roaring bitmaps in a group calculated based on UNION.

Example:

```
SELECT a, rb_or_cardinality_agg(b) FROM r1 GROUP BY a ORDER BY 1;
```

| a | rb_or_cardinality_agg |
|----|-----------------------|
| 1 | 1 |
| 2 | 2 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 2 |
| 10 | 2 |

(10 rows)

rb_xor_cardinality_agg(roaringbitmap)

Description: Cardinality of Roaring bitmaps in a group calculated based on XOR.

Example:

```
SELECT a, rb_xor_cardinality_agg(b) FROM r1 GROUP BY a ORDER BY 1;
a | rb_xor_cardinality_agg
-----+-----
1 | 0
2 | 1
3 | 1
4 | 1
5 | 1
6 | 1
7 | 1
8 | 1
9 | 1
10 | 1
(10 rows)
```

21.7.19.4 Use Cases

Background

Currently, real-time precision marketing is required in the Internet, education, and gaming industries. User profiling enables user search based on combined criteria. Example:

- Before launching a sales promotion, e-commerce companies need to send the promotion message to a selected batch of users with specific features.
- In education, exercise questions need to be pushed based on students' weaknesses.
- On search, video, and portal websites, different contents are pushed to users based on their interests.

These use cases have the following characteristics in common:

- The data volume is huge and the calculation workload is huge.
- There are a large number of users with a lot of labels and fields, occupying a large amount of storage space.
- The feature conditions for selection are diversified, and it is difficult to find a fixed index. If each field has an index, that will occupy too much storage space.
- High performance is required because real-time marketing requires response in seconds.
- Data update has high requirements on timeliness, and user profiles need to be updated in real time.

Roaring bitmaps in DWS can efficiently generate, compress, and parse bitmap data, and supports the most common bitmap aggregation operations (AND, OR, NOT, and XOR). This feature meets the requirements of real-time precision marketing and quick user selection in the case of a large amount of data with hundreds of millions of users and tens of millions of labels.

Using roaringbitmap

Assume that there is a web page browsing information table **userinfo**. The fields in the table are as follows:

```
CREATE TABLE userinfo
(userid int,
age int,
```



```
gender text,  
salary int,  
hobby text  
)WITH (orientation=column);
```

The data in the **userinfo** table increases with the change of user information. For example, if a user has multiple hobby attributes, there will be multiple records in the **userinfo** table.

If a user wants to filter out males with income greater than CNY10,000, age greater than 30, and a hobby of phishing, and then push specific messages to these target groups.

The traditional method is to directly query the original table. The statement is as follows:

```
SELECT distinct userid FROM userinfo WHERE salary > 10000 AND age > 30 AND gender ='m' AND hobby  
='fishing';
```

If the **userinfo** table contains a small amount of data, indexes are created in the **salary**, **age**, **gender**, and **hobby** columns to meet the query requirements. However, if the **userinfo** table contains a large amount of data and a large number of labels, the preceding statement cannot meet the requirements. The reasons are as follows:

- A large number of indexes need to be created.
- The count (distinct) performance is poor.

Roaring bitmaps perform better in this case.

1. Create a RoaringBitmap table.

```
CREATE TABLE userinfoet  
( age int,  
gender text,  
salary int,  
hobby text,  
userset roaringbitmap,  
PRIMARY KEY(age,gender,salary,hobby)  
)WITH (orientation=column);
```

2. All data in the **userinfo** table must be aggregated to the **userinfoet** table through the aggregation of the label column. You can run the following command to aggregate all data. Or you can aggregate only incremental data. To aggregate only incremental data, a set of users with the same label are put in a table record. This can be implemented by using the UPSERT function. Frequent update operations may generate a large amount of dirty data. Therefore, you are advised to create the **userinfoet** table as a row-store table to aggregate incremental data.

```
INSERT INTO userinfoet  
SELECT age, gender, salary, hobby, rb_build_agg(userid)  
FROM  
userinfo  
GROUP BY age, gender, salary, hobby;
```

3. Query the **userinfoet** table for the selected user information.

```
SELECT rb_iterate(rb_or_agg(userset)) FROM userinfoet WHERE salary > 10000 AND age > 30 AND  
gender ='m' AND hobby ='fishing';
```

After data aggregation, the data volume of the table **userinfoet** is much smaller than that of the source table, so the scanning performance of the base table is much faster. In addition, based on the advantages of Roaring bitmaps, the performance of calculating **rb_or_agg** and **rb_iterate** is better. Compared with the traditional method, the performance is significantly improved.

21.7.20 UUID Functions

UUID functions are used to generate UUID data (see [UUID Type](#)).

uuid_generate_v1()

Description: Generates a UUID sequence number.

Return type: UUID

Example:

```
SELECT uuid_generate_v1();
       uuid_generate_v1
-----
c71ceaca-a175-11e9-a920-797ff7000001
(1 row)
```

NOTE

The **uuid_generate_v1** function generates a UUID based on the time information, cluster node ID, and ID of the thread that generates the sequence. The UUID is globally unique in a single cluster, however, there is a possibility that the time information, cluster node IDs, thread IDs, and clock sequences of multiple clusters are the same at the same time. Therefore, there is a low probability that UUIDs generated among multiple clusters are duplicate.

uuid()

Description: Generates a UUID sequence number. This function is a MySQL compatibility function supported only by version 8.2.0 and later clusters.

Return type: UUID

Example:

```
SELECT uuid();
       uuid
-----
6327dc96-f0e7-0100-f2f2-6c9ff700fffe
(1 row)
```

NOTE

UUID is generated in the way the **uuid_generate_v1()** function works. The function generates a UUID based on the time information, cluster node ID, and ID of the thread that generates the sequence. The UUID is globally unique in a single cluster, however, there is a possibility that the time information, cluster node IDs, thread IDs, and clock sequences of multiple clusters are the same at the same time. Therefore, there is a low probability that UUIDs generated among multiple clusters are duplicate.

sys_guid()

Description: Generates an Oracle GUID, which is similar to the UUID. This function is compatible with Oracle.

Return type: text

Example:

```
SELECT sys_guid();
       sys_guid
-----
```

```
-----  
4EBD3C74A17A11E9A1BF797FF7000001  
(1 row)
```

NOTE

The data generation principle of the `sys_guid` function is the same as that of the `uuid_generate_v1` function.

UUID Function Application Example

A UUID is globally unique. It can be used as a primary key or a distribution column in a data table. When **`uuid_generate_v1()`** is used as the default value of the distribution column in a data table, data can be evenly distributed to each DN through hash distribution to prevent data skew.

NOTE

The obvious advantage of UUID is that it is globally unique and does not require a central node. UUIDs are generated independently on a single node. However, UUIDs occupy more storage space than INTs, the index efficiency is low, and the generated IDs are random, making them difficult to identify. Therefore, you need to select UUID or Sequence as the primary key of the data table based on the site requirements.

An example is as follows:

1. The INT type is used as the distribution column.

An example hash table **`mytable01`** is created, and the ints are used as the distribution column. After data is inserted, data skew occurs.

```
CREATE TABLE mytable01(a INT, b INT) DISTRIBUTE BY hash(a);  
CREATE TABLE  
INSERT INTO mytable01 VALUES(1, 10);  
INSERT 0 1  
INSERT INTO mytable01 VALUES(1, 10);  
INSERT 0 1  
INSERT INTO mytable01 VALUES(1, 10);  
INSERT 0 1  
INSERT INTO mytable01 VALUES(1, 10);  
INSERT 0 1  
INSERT INTO mytable01 VALUES(1, 10);  
INSERT 0 1  
SELECT * FROM mytable01;  
a | b  
---+---  
1 | 10  
1 | 10  
1 | 10  
1 | 10  
1 | 10  
(5 rows)  
  
SELECT table_skewness('mytable01');  
table_skewness  
-----  
("dn_6003_6004      ",5,100.000%)  
("dn_6001_6002      ",0,0.000%)  
("dn_6005_6006      ",0,0.000%)  
(3 rows)
```

2. The UUIDs are used as the distribution column.

Create an example hash table **`mytable02`** and use the UUIDs as the distribution column. After data is inserted, data distribution is normal.

```
CREATE TABLE mytable02(id UUID default uuid_generate_v1(), a INT, b INT) DISTRIBUTE BY hash(id);  
CREATE TABLE
```

```
INSERT INTO mytable02(a, b) VALUES(1, 10);
INSERT 0 1
INSERT INTO mytable02(a, b) VALUES(1, 10);
INSERT 0 1
INSERT INTO mytable02(a, b) VALUES(1, 10);
INSERT 0 1
INSERT INTO mytable02(a, b) VALUES(1, 10);
INSERT 0 1
INSERT INTO mytable02(a, b) VALUES(1, 10);
INSERT 0 1

SELECT * FROM mytable02;
      id      | a | b
-----+---+---
63e45c14-cc74-0e00-e9aa-0a2c3fa0fffe | 1 | 10
63e45c1f-4d18-0700-e9ab-0a2c3fa0fffe | 1 | 10
63e45c26-f859-0b00-e9ad-0a2c3fa0fffe | 1 | 10
63e45c23-9e5d-0300-e9ac-0a2c3fa0fffe | 1 | 10
63e45c2a-5825-0600-e9ae-0a2c3fa0fffe | 1 | 10
(5 rows)

SELECT table_skewness('mytable02');
      table_skewness
-----
("dn_6001_6002      ",3,60.000%)
("dn_6003_6004      ",2,40.000%)
("dn_6005_6006      ",0,0.000%)
(3 rows)
```

21.7.21 Text Search Functions and Operators

21.7.21.1 Text Search Operators

@@

Description: Specifies whether the **tsvector**-typed words match the **tsquery**-typed words.

Example:

```
SELECT to_tsvector('fat cats ate rats') @@ to_tsquery('cat & rat') AS RESULT;
result
-----
t
(1 row)
```

@@@

Description: Synonym for @@

Example:

```
SELECT to_tsvector('fat cats ate rats') @@@ to_tsquery('cat & rat') AS RESULT;
result
-----
t
(1 row)
```

&&

Description: Performs the AND operation on two **tsquery**-typed words.

Example:

```
SELECT 'fat | rat'::tsquery && 'cat'::tsquery AS RESULT;  
result  
-----  
( 'fat' | 'rat' ) & 'cat'  
(1 row)
```

||

Description: Performs the OR operation on two **tsquery**-typed words.

Example:

```
SELECT 'fat | rat'::tsquery || 'cat'::tsquery AS RESULT;  
result  
-----  
( 'fat' | 'rat' ) | 'cat'  
(1 row)  
SELECT 'a:1 b:2'::tsvector || 'c:1 d:2 b:3'::tsvector AS RESULT;  
result  
-----  
'a':1 'b':2,5 'c':3 'd':4  
(1 row)
```

!!

Description: **NOT** a **tsquery**

Example:

```
SELECT !! 'cat'::tsquery AS RESULT;  
result  
-----  
!'cat'  
(1 row)
```

@>

Description: Specifies whether a **tsquery**-typed word contains another **tsquery**-typed word.

Example:

```
SELECT 'cat'::tsquery @> 'cat & rat'::tsquery AS RESULT;  
result  
-----  
f  
(1 row)
```

<@

Description: Specifies whether a **tsquery**-typed word is contained in another **tsquery**-typed word.

Example:

```
SELECT 'cat'::tsquery <@ 'cat & rat'::tsquery AS RESULT;  
result  
-----  
t  
(1 row)
```

In addition to the preceding operators, the ordinary B-tree comparison operators (including = and <) are defined for types **tsvector** and **tsquery**.

21.7.21.2 Text Search Functions

get_current_ts_config()

Description: Gets default text search configuration.

Return type: regconfig

Example:

```
SELECT get_current_ts_config();
get_current_ts_config
-----
english
(1 row)
```

length(tsvector)

Description: Number of lexemes in a **tsvector**-typed word.

Return type: integer

Example:

```
SELECT length('fat:2,4 cat:3 rat:5A'::tsvector);
length
-----
3
(1 row)
```

numnode(tsquery)

Description: Number of lexemes plus **tsquery** operators

Return type: integer

Example:

```
SELECT numnode('(fat & rat) | cat'::tsquery);
numnode
-----
5
(1 row)
```

plainto_tsquery([config regconfig ,] query text)

Description: Generates **tsquery** lexemes without punctuation.

Return type: tsquery

Example:

```
SELECT plainto_tsquery('english', 'The Fat Rats');
plainto_tsquery
-----
'fat' & 'rat'
(1 row)
```

querytree(query tsquery)

Description: Gets indexable part of a **tsquery**.

Return type: text

Example:

```
SELECT querytree('foo & ! bar'::tsquery);
querytree
-----
'foo'
(1 row)
```

setweight(tsvector, "char")

Description: Assigns weight to each element of **tsvector**.

Return type: tsvector

Example:

```
SELECT setweight('fat:2,4 cat:3 rat:5B'::tsvector, 'A');
setweight
-----
'cat':3A 'fat':2A,4A 'rat':5A
(1 row)
```

strip(tsvector)

Description: Removes positions and weights from **tsvector**.

Return type: tsvector

Example:

```
SELECT strip('fat:2,4 cat:3 rat:5A'::tsvector);
strip
-----
'cat' 'fat' 'rat'
(1 row)
```

to_tsquery([config regconfig ,] query text)

Description: Normalizes words and converts them to **tsquery**.

Return type: tsquery

Example:

```
SELECT to_tsquery('english', 'The & Fat & Rats');
to_tsquery
-----
'fat' & 'rat'
(1 row)
```

to_tsvector([config regconfig ,] document text)

Description: Reduces document text to **tsvector**.

Return type: tsvector

Example:

```
SELECT to_tsvector('english', 'The Fat Rats');
       to_tsvector
-----
'fat':2 'rat':3
(1 row)
```

ts_headline([config regconfig,] document text, query tsquery [, options text])

Description: Highlights a query match.

Return type: text

Example:

```
SELECT ts_headline('x y z', 'z'::tsquery);
       ts_headline
-----
x y <b>z</b>
(1 row)
```

ts_rank([weights float4[],] vector tsvector, query tsquery [, normalization integer])

Description: Ranks document for query.

Return type: float4

Example:

```
SELECT ts_rank('hello world'::tsvector, 'world'::tsquery);
       ts_rank
-----
.0607927
(1 row)
```

ts_rank_cd([weights float4[],] vector tsvector, query tsquery [, normalization integer])

Description: Ranks document for query using cover density.

Return type: float4

Example:

```
SELECT ts_rank_cd('hello world'::tsvector, 'world'::tsquery);
       ts_rank_cd
-----
0
(1 row)
```

ts_rewrite(query tsquery, target tsquery, substitute tsquery)

Description: Replaces **tsquery**-typed word.

Return type: tsquery

Example:

```
SELECT ts_rewrite('a & b'::tsquery, 'a'::tsquery, 'foo|bar'::tsquery);
       ts_rewrite
-----
```



```
'b' & ( 'foo' | 'bar' )  
(1 row)
```

ts_rewrite(query tsquery, select text)

Description: Replaces **tsquery** data in the target with the result of a **SELECT** command.

Return type: tsquery

Example:

```
SELECT ts_rewrite('world'::tsquery, 'select "world"::tsquery, "hello"::tsquery');  
ts_rewrite  
-----  
'hello'  
(1 row)
```

21.7.21.3 Text Search Debugging Functions

ts_debug([config regconfig,] document text, OUT alias text, OUT description text, OUT token text, OUT dictionaries regdictionary[], OUT dictionary regdictionary, OUT lexemes text[])

Description: Tests a configuration.

Return type: SETOF record

Example:

```
SELECT ts_debug('english', 'The Brightest supernovae');  
ts_debug  
-----  
(asciiword,"Word, all ASCII",The,{english_stem},english_stem,{})  
(blank,"Space symbols"," ",{},{})  
(asciiword,"Word, all ASCII",Brightest,{english_stem},english_stem,{brightest})  
(blank,"Space symbols"," ",{},{})  
(asciiword,"Word, all ASCII",supernovae,{english_stem},english_stem,{supernova})  
(5 rows)
```

ts_lexize(dict regdictionary, token text)

Description: Tests a data dictionary.

Return type: text[]

Example:

```
SELECT ts_lexize('english_stem', 'stars');  
ts_lexize  
-----  
{star}  
(1 row)
```

ts_parse(parser_name text, document text, OUT tokid integer, OUT token text)

Description: Tests a parser.

Return type: SETOF record

Example:

```
SELECT ts_parse('default', 'foo - bar');
ts_parse
-----
(1,foo)
(12," ")
(12,"- ")
(1,bar)
(4 rows)
```

ts_parse(parser_oid oid, document text, OUT tokid integer, OUT token text)

Description: Tests a parser.

Return type: SETOF record

Example:

```
SELECT ts_parse(3722, 'foo - bar');
ts_parse
-----
(1,foo)
(12," ")
(12,"- ")
(1,bar)
(4 rows)
```

ts_token_type(parser_name text, OUT tokid integer, OUT alias text, OUT description text)

Description: Gets token types defined by parser.

Return type: SETOF record

Example:

```
SELECT ts_token_type('default');
ts_token_type
-----
(1,asciiword,"Word, all ASCII")
(2,word,"Word, all letters")
(3,numword,"Word, letters and digits")
(4,email,"Email address")
(5,url,URL)
(6,host,Host)
(7,sfloat,"Scientific notation")
(8,version,"Version number")
(9,hword_numpart,"Hyphenated word part, letters and digits")
(10,hword_part,"Hyphenated word part, all letters")
(11,hword_asciipart,"Hyphenated word part, all ASCII")
(12,blank,"Space symbols")
(13,tag,"XML tag")
(14,protocol,"Protocol head")
(15,numhword,"Hyphenated word, letters and digits")
(16,asciihword,"Hyphenated word, all ASCII")
(17,hword,"Hyphenated word, all letters")
(18,url_path,"URL path")
(19,file,"File or path name")
(20,float,"Decimal notation")
(21,int,"Signed integer")
(22,uint,"Unsigned integer")
(23,entity,"XML entity")
(23 rows)
```

ts_token_type(parser_oid oid, OUT tokid integer, OUT alias text, OUT description text)

Description: Gets token types defined by parser.

Return type: SETOF record

Example:

```
SELECT ts_token_type(3722);
      ts_token_type
-----
(1,asciiword,"Word, all ASCII")
(2,word,"Word, all letters")
(3,numword,"Word, letters and digits")
(4,email,"Email address")
(5,url,URL)
(6,host,Host)
(7,sfloat,"Scientific notation")
(8,version,"Version number")
(9,hword_numpart,"Hyphenated word part, letters and digits")
(10,hword_part,"Hyphenated word part, all letters")
(11,hword_asciipart,"Hyphenated word part, all ASCII")
(12,blank,"Space symbols")
(13,tag,"XML tag")
(14,protocol,"Protocol head")
(15,numhword,"Hyphenated word, letters and digits")
(16,asciihword,"Hyphenated word, all ASCII")
(17,hword,"Hyphenated word, all letters")
(18,url_path,"URL path")
(19,file,"File or path name")
(20,float,"Decimal notation")
(21,int,"Signed integer")
(22,uint,"Unsigned integer")
(23,entity,"XML entity")
(23 rows)
```

ts_stat(sqlquery text, [weights text,] OUT word text, OUT ndoc integer, OUT nentry integer)

Description: Gets statistics of a **tsvector** column.

Return type: SETOF record

Example:

```
SELECT ts_stat('select "hello world"::tsvector');
      ts_stat
-----
(world,1,1)
(hello,1,1)
(2 rows)
```

21.7.22 HLL Functions and Operators

21.7.22.1 HLL Operators

The HLL type supports the following operators:

Table 21-48 HLL Operators

| HLL Operators | Description | Return Type | Example |
|---------------|--|-------------|---|
| = | Checks whether the values of hll and hll_hashval are equal. | bool | <ul style="list-style-type: none"> hll
 <pre>SELECT (hll_empty() hll_hash_integer(1)) = (hll_empty() hll_hash_integer(1)); ?column?</pre> <pre>----- t (1 row)</pre> hll_hashval
 <pre>SELECT hll_hash_integer(1) = hll_hash_integer(1); ?column?</pre> <pre>----- t (1 row)</pre> |
| <> or != | Checks whether the values of hll and hll_hashval are not equal. | bool | <ul style="list-style-type: none"> hll
 <pre>SELECT (hll_empty() hll_hash_integer(1)) <> (hll_empty() hll_hash_integer(2)); ?column?</pre> <pre>----- t (1 row)</pre> hll_hashval
 <pre>SELECT hll_hash_integer(1) <> hll_hash_integer(2); ?column?</pre> <pre>----- t (1 row)</pre> |
| | Represents the functions of hll_add , hll_union , and hll_add_rev . | hll | <ul style="list-style-type: none"> hll_add
 <pre>SELECT hll_empty() hll_hash_integer(1); ?column?</pre> <pre>----- \x128b7f8895a3f5af28cafe (1 row)</pre> hll_add_rev
 <pre>SELECT hll_hash_integer(1) hll_empty(); ?column?</pre> <pre>----- \x128b7f8895a3f5af28cafe (1 row)</pre> hll_union
 <pre>SELECT (hll_empty() hll_hash_integer(1)) (hll_empty() hll_hash_integer(2)); ?column?</pre> <pre>----- \x128b7f8895a3f5af28cafeda0ce907e4355b60 (1 row)</pre> |

| HLL Operators | Description | Return Type | Example |
|---------------|---|-------------|---|
| # | Calculates the distinct value of the hll , which is equivalent to the hll_cardinality function. | integer | <pre>SELECT #(hll_empty() hll_hash_integer(1)); ?column? ----- 1 (1 row)</pre> |

21.7.22.2 Hash Functions

hll_hash_boolean(bool)

Description: Hashes data of the bool type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_boolean(FALSE);
hll_hash_boolean
-----
5048724184180415669
(1 row)
```

hll_hash_boolean(bool, int32)

Description: Configures a hash seed (that is, change the hash policy) and hashes data of the bool type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_boolean(FALSE, 10);
hll_hash_boolean
-----
391264977436098630
(1 row)
```

hll_hash_smallint(smallint)

Description: Hashes data of the smallint type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_smallint(100::smallint);
hll_hash_smallint
-----
4631120266694327276
(1 row)
```

 NOTE

If parameters with the same numeric value are hashed using different data types, the data will differ, because hash functions select different calculation policies for each type.

hll_hash_smallint(smallint, int32)

Description: Configures a hash seed (that is, change the hash policy) and hashes data of the smallint type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_smallint(100::smallint, 10);
 hll_hash_smallint
-----
8349353095166695771
(1 row)
```

hll_hash_integer(integer)

Description: Hashes data of the integer type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_integer(0);
 hll_hash_integer
-----
-3485513579396041028
(1 row)
```

hll_hash_integer(integer, int32)

Description: Hashes data of the integer type and configures a hash seed (that is, change the hash policy).

Return type: hll_hashval

Example:

```
SELECT hll_hash_integer(0, 10);
 hll_hash_integer
-----
183371090322255134
(1 row)
```

hll_hash_bigint(bigint)

Description: Hashes data of the bigint type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_bigint(100::bigint);
 hll_hash_bigint
-----
8349353095166695771
(1 row)
```

hll_hash_bigint(bigint, int32)

Description: Hashes data of the bigint type and configures a hash seed (that is, change the hash policy).

Return type: hll_hashval

Example:

```
SELECT hll_hash_bigint(100::bigint, 10);
 hll_hash_bigint
-----
4631120266694327276
(1 row)
```

hll_hash_bytea(bytea)

Description: Hashes data of the bytea type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_bytea(E'\\x');
 hll_hash_bytea
-----
0
(1 row)
```

hll_hash_bytea(bytea, int32)

Description: Hashes data of the bytea type and configures a hash seed (that is, change the hash policy).

Return type: hll_hashval

Example:

```
SELECT hll_hash_bytea(E'\\x', 10);
 hll_hash_bytea
-----
6574525721897061910
(1 row)
```

hll_hash_text(text)

Description: Hashes data of the text type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_text('AB');
 hll_hash_text
-----
5365230931951287672
(1 row)
```

hll_hash_text(text, int32)

Description: Hashes data of the text type and configures a hash seed (that is, change the hash policy).

Return type: hll_hashval

Example:

```
SELECT hll_hash_text('AB', 10);
       hll_hash_text
-----
7680762839921155903
(1 row)
```

hll_hash_any(anytype)

Description: Hashes data of any type.

Return type: hll_hashval

Example:

```
SELECT hll_hash_any(1);
       hll_hash_any
-----
-8604791237420463362
(1 row)

SELECT hll_hash_any('08:00:2b:01:02:03'::macaddr);
       hll_hash_any
-----
-4883882473551067169
(1 row)
```

hll_hash_any(anytype, int32)

Description: Hashes data of any type and configures a hash seed (that is, change the hash policy).

Return type: hll_hashval

Example:

```
SELECT hll_hash_any(1, 10);
       hll_hash_any
-----
-1478847531811254870
(1 row)
```

hll_hashval_eq(hll_hashval, hll_hashval)

Description: Compares two pieces of data of the hll_hashval type to check whether they are the same.

Return type: bool

Example:

```
SELECT hll_hashval_eq(hll_hash_integer(1), hll_hash_integer(1));
       hll_hashval_eq
-----
t
(1 row)
```

hll_hashval_ne(hll_hashval, hll_hashval)

Description: Compares two pieces of data of the hll_hashval type to check whether they are different.

Return type: bool

Example:

```
SELECT hll_hashval_ne(hll_hash_integer(1), hll_hash_integer(1));
hll_hashval_ne
-----
f
(1 row)
```

21.7.22.3 Precision Functions

HLL supports **explicit**, **sparse**, and **full** modes. **explicit** and **sparse** excel when the data scale is small, and barely produce errors in calculation results. When the number of distinct values increases, **full** becomes more suitable, but produces some errors. The following functions are used to view precision parameters in HLLs.

hll_schema_version(hll)

Description: Checks the schema version in the current HLL.

Return type: integer

Example:

```
SELECT hll_schema_version(hll_empty());
hll_schema_version
-----
1
(1 row)
```

hll_type(hll)

Description: Checks the type of the current HLL.

Return type: integer

Example:

```
SELECT hll_type(hll_empty());
hll_type
-----
1
(1 row)
```

hll_log2m(hll)

Description: Check the value of log2m of the current HLL. This value affects the error rate in calculating the number of distinct values by the HLL. The formula for calculating the error rate is as follows:

$$\pm 1.04 / \sqrt{2^{\wedge} \log 2m}$$

Return type: integer

Example:

```
SELECT hll_log2m(hll_empty());
hll_log2m
-----
```

```
11  
(1 row)
```

hll_regwidth(hll)

Description: Checks the number of bits of buckets in a hll data structure.

Return type: integer

Example:

```
SELECT hll_regwidth(hll_empty());  
hll_regwidth  
-----  
5  
(1 row)
```

hll_expthresh(hll)

Description: Obtains the size of **expthresh** in the current HLL. An HLL usually switches from the **explicit** mode to the **sparse** mode and then to the **full** mode. This process is called the promotion hierarchy policy. You can change the value of **expthresh** to change the policy. For example, if **expthresh** is **0**, an HLL will skip the **explicit** mode and directly enter the **sparse** mode. If the value of **expthresh** is explicitly set to a value ranging from 1 to 7, this function returns $2^{\text{expthresh}}$.

Return type: record

Example:

```
SELECT hll_expthresh(hll_empty());  
hll_expthresh  
-----  
(-1,160)  
(1 row)  
  
SELECT hll_expthresh(hll_empty(11,5,3));  
hll_expthresh  
-----  
(8,8)  
(1 row)
```

hll_sparseon(hll)

Description: Specifies whether to enable the **sparse** mode. **0** indicates **off** and **1** indicates **on**.

Return type: integer

Example:

```
SELECT hll_sparseon(hll_empty());  
hll_sparseon  
-----  
1  
(1 row)
```

21.7.22.4 Aggregate Functions

hll_add_agg(hll_hashval)

Description: Groups hashed data into HLL.

Return type: hll

Example:

1. Prepare data.

```
CREATE TABLE t_id(id int);
INSERT INTO t_id VALUES(generate_series(1,500));
CREATE TABLE t_data(a int, c text);
INSERT INTO t_data SELECT mod(id,2), id FROM t_id;
```
2. Create another table and specify an HLL column:

```
CREATE TABLE t_a_c_hll(a int, c hll);
```
3. Use **GROUP BY** on column **a** to group data, and insert the data to the HLL column:

```
INSERT INTO t_a_c_hll SELECT a, hll_add_agg(hll_hash_text(c)) FROM t_data GROUP BY a;
```
4. Calculate the number of distinct values for each group in the HLL column:

```
SELECT a, #c as cardinality FROM t_a_c_hll order by a;
```

| a | cardinality |
|---|------------------|
| 0 | 250.741759091658 |
| 1 | 250.741759091658 |

(2 rows)

hll_add_agg(hll_hashval, int32 log2m)

Description: Groups hashed data into HLL and sets the **log2m** parameter. The parameter value ranges from 10 to 16.

Return type: hll

Example:

```
SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), 10)) FROM t_data;
```

| hll_cardinality |
|------------------|
| 503.932348927339 |

(1 row)

hll_add_agg(hll_hashval, int32 log2m, int32 regwidth)

Description: Groups hashed data into HLL and sets the **log2m** and **regwidth** parameters in sequence. The value of **regwidth** ranges from 1 to 5.

Return type: hll

Example:

```
SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1)) FROM t_data;
```

| hll_cardinality |
|------------------|
| 496.628982624022 |

(1 row)

hll_add_agg(hll_hashval, int32 log2m, int32 regwidth, int64 expthresh)

Description: Groups hashed data into **hll** and sets parameters **log2m**, **regwidth**, and **expthresh** in sequence. The value of **expthresh** is an integer ranging from -1 to 7. **expthresh** is used to specify the threshold for switching from the **explicit** mode to the **sparse** mode. -1 indicates the auto mode; 0 indicates that the **explicit** mode is skipped; a value from 1 to 7 indicates that the mode is switched when the number of distinct values reaches $2^{\text{expthresh}}$.

Return type: hll

Example:

```
SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1, 4)) FROM t_data;  
hll_cardinality  
-----  
496.628982624022  
(1 row)
```

hll_add_agg(hll_hashval, int32 log2m, int32 regwidth, int64 expthresh, int32 sparseon)

Description: Groups hashed data into **hll** and sets parameters **log2m**, **regwidth**, **expthresh**, and **sparseon** in sequence. The value of **sparseon** is 0 or 1.

Return type: hll

Example:

```
SELECT hll_cardinality(hll_add_agg(hll_hash_text(c), NULL, 1, 4, 0)) FROM t_data;  
hll_cardinality  
-----  
496.628982624022  
(1 row)
```

hll_union_agg(hll)

Description: Perform the **UNION** operation on multiple pieces of data of the hll type to obtain one HLL.

Return type: hll

Example:

Perform the **UNION** operation on data of the HLL type in each group to obtain one HLL, and calculate the number of distinct values:

```
SELECT #hll_union_agg(c) as cardinality FROM t_a_c_hll;  
cardinality  
-----  
496.628982624022  
(1 row)
```

NOTE

To perform **UNION** on data in multiple HLLs, ensure that the HLLs have the same precision. Otherwise, **UNION** cannot be performed. This restriction also applies to the `hll_union(hll, hll)` function.

21.7.22.5 Functional Functions

hll_print(hll)

Description: Prints some debugging parameters of an HLL.

Return type: cstring

Example:

```
SELECT hll_print(hll_empty());  
hll_print
```

```
-----  
EMPTY, nregs=2048, nbits=5, expthresh=-1(160), sparseon=1  
(1 row)
```

hll_empty()

Description: Creates an empty HLL.

Return type: hll

Example:

```
SELECT hll_empty();  
hll_empty  
-----  
\x118b7f  
(1 row)
```

hll_empty(int32 log2m)

Description: Creates an empty HLL and sets the **log2m** parameter. The parameter value ranges from 10 to 16.

Return type: hll

Example:

```
SELECT hll_empty(10);  
hll_empty  
-----  
\x118a7f  
(1 row)
```

hll_empty(int32 log2m, int32 regwidth)

Description: Creates an empty HLL and sets the **log2m** and **regwidth** parameters in sequence. The value of **regwidth** ranges from 1 to 5.

Return type: hll

Example:

```
SELECT hll_empty(10, 4);  
hll_empty  
-----  
\x116a7f  
(1 row)
```

hll_empty(int32 log2m, int32 regwidth, int64 expthresh)

Description: Creates an empty HLL and sets the **log2m**, **regwidth**, and **expthresh** parameters. The value of **expthresh** is an integer ranging from -1 to 7. This parameter specifies the threshold for switching from the **explicit** mode to the **sparse** mode. -1 indicates the auto mode; 0 indicates that the **explicit** mode is skipped; a value from 1 to 7 indicates that the mode is switched when the number of distinct values reaches $2^{\text{expthresh}}$.

Return type: hll

Example:

```
SELECT hll_empty(10, 4, 7);  
hll_empty
```

```
-----  
\x116a48  
(1 row)
```

hll_empty(int32 log2m, int32 regwidth, int64 expthresh, int32 sparseon)

Description: Creates an empty HLL and sets the **log2m**, **regwidth**, **expthresh**, and **sparseon** parameters. The value of **sparseon** is 0 or 1.

Return type: hll

Example:

```
SELECT hll_empty(10,4,7,0);  
hll_empty  
-----  
\x116a08  
(1 row)
```

hll_add(hll, hll_hashval)

Description: Adds hll_hashval to an HLL.

Return type: hll

Example:

```
SELECT hll_add(hll_empty(), hll_hash_integer(1));  
hll_add  
-----  
\x128b7f8895a3f5af28cafe  
(1 row)
```

hll_add_rev(hll_hashval, hll)

Description: Adds hll_hashval to an HLL. This function works the same as hll_add, except that the positions of parameters are switched.

Return type: hll

Example:

```
SELECT hll_add_rev(hll_hash_integer(1), hll_empty());  
hll_add_rev  
-----  
\x128b7f8895a3f5af28cafe  
(1 row)
```

hll_eq(hll, hll)

Description: Compares two HLLs to check whether they are the same.

Return type: bool

Example:

```
SELECT hll_eq(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(), hll_hash_integer(2)));  
hll_eq  
-----  
f  
(1 row)
```

hll_ne(hll, hll)

Description: Compares two HLLs to check whether they are different.

Return type: bool

Example:

```
SELECT hll_ne(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(), hll_hash_integer(2)));
hll_ne
-----
t
(1 row)
```

hll_cardinality(hll)

Description: Calculates the number of distinct values of an HLL.

Return type: integer

Example:

```
SELECT hll_cardinality(hll_empty() || hll_hash_integer(1));
hll_cardinality
-----
1
(1 row)
```

hll_union(hll, hll)

Description: Performs the **UNION** operation on two HLL data structures to obtain one HLL.

Return type: hll

Example:

```
SELECT hll_union(hll_add(hll_empty(), hll_hash_integer(1)), hll_add(hll_empty(), hll_hash_integer(2)));
hll_union
-----
\x128b7f8895a3f5af28cafeda0ce907e4355b60
(1 row)
```

21.7.22.6 Built-in Functions

HyperLogLog (HLL) has a series of built-in functions for internal data processing. Generally, you are not advised to use these functions.

Table 21-49 HLL built-in functions

| Function | Function |
|--------------|--|
| hll_in | Receives hll data in string format. |
| hll_out | Sends hll data in string format. |
| hll_rcv | Receives hll data in bytea format. |
| hll_send | Sends hll data in bytea format. |
| hll_trans_in | Receives hll_trans_type data in string format. |

| Function | Function |
|-------------------|---|
| hll_trans_out | Sends hll_trans_type data in string format. |
| hll_trans_recv | Receives hll_trans_type data in bytea format. |
| hll_trans_send | Sends hll_trans_type data in bytea format. |
| hll_tymod_in | Receives tymod data. |
| hll_tymod_out | Sends tymod data. |
| hll_hashval_in | Receives hll_hashval data. |
| hll_hashval_out | Sends hll_hashval data. |
| hll_add_trans0 | Works similar to hll_add , and is used during the initial phase of DNs in distributed aggregation procedures. |
| hll_union_trans | Works similar to hll_union , and is used on the first phase of DNs in distributed aggregation operations. |
| hll_union_collect | Works similar to hll_union , and is used on the second phase of CNs in distributed aggregation operations to summarize the results of each DN. |
| hll_pack | Is used on the third phase of CNs in distributed aggregation operations to convert a user-defined type hll_trans_type to the hll type. |
| hll | Converts a hll type to another hll type. Input parameters can be specified. |
| hll_hashval | Converts the bigint type to the hll_hashval type. |
| hll_hashval_int4 | Converts the int4 type to the hll_hashval type. |

21.7.23 Set Returning Functions

21.7.23.1 Series Generating Functions

generate_series() returns a series-based set based on the specified start value (**start**), end value (**stop**), and step (**step**).

If **step** is a positive number and **start** is greater than **stop**, zero row is returned. If **step** is a negative number and **start** is less than **stop**, zero row is returned. If any input is NULL, zero rows are returned. If the value of **step** is **0**, an error is reported.

generate_series(start, stop)

Description: Generates a series of values, from **start** to **stop**, the default step is **1**.

Parameter type: int, bigint, or numeric

Return type: setof int, setof bigint, or setof numeric (same as the argument type)

Example:

```
SELECT * FROM generate_series(2,4);
generate_series
-----
         2
         3
         4
(3 rows)

SELECT * FROM generate_series(4,3);
generate_series
-----
(0 rows)

SELECT * FROM generate_series(1,NULL);
generate_series
-----
(0 rows)
```

generate_series(start, stop, step)

Description: Generates a series of values, from **start** to **stop** with a step size of **step**.

Parameter type: int, bigint, or numeric

Return type: setof int, setof bigint, or setof numeric (same as the argument type)

Example:

```
SELECT * FROM generate_series(5,1,-2);
generate_series
-----
         5
         3
         1
(3 rows)

SELECT * FROM generate_series(4,6,-5);
generate_series
-----
(0 rows)

SELECT * FROM generate_series(4,3,0);
ERROR: step size cannot equal zero
```

generate_series(start, stop, step interval)

Description: Generates a series of values, from **start** to **stop** with a step size of **step**.

Parameter type: timestamp or timestamp with time zone

Return type: setof timestamp or setof timestamp with time zone (same as argument type)

Example:

```
-- this example relies on the date-plus-integer operator
SELECT current_date + s.a AS dates FROM generate_series(0,14,7) AS s(a);
-----
2017-06-02
```

```
2017-06-09
2017-06-16
(3 rows)

SELECT * FROM generate_series('2008-03-01 00:00'::timestamp, '2008-03-04 12:00', '10 hours');
generate_series
-----
2008-03-01 00:00:00
2008-03-01 10:00:00
2008-03-01 20:00:00
2008-03-02 06:00:00
2008-03-02 16:00:00
2008-03-03 02:00:00
2008-03-03 12:00:00
2008-03-03 22:00:00
2008-03-04 08:00:00
(9 rows)
```

21.7.23.2 Subscript Generating Functions

generate_subscripts(array anyarray, dim int)

Description: Generates a series comprising the given array's subscripts.

Return type: setof int

Example:

```
SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1) AS s;
s
---
1
2
3
4
(4 rows)
```

generate_subscripts(array anyarray, dim int, reverse boolean)

Description: Generates a series comprising the given array's subscripts. When **reverse** is true, the series is returned in reverse order.

Return type: setof int

Example:

```
SELECT generate_subscripts('{NULL,1,NULL,2}'::int[], 1,TRUE) AS s;
s
---
4
3
2
1
(4 rows)
```

generate_subscripts is a function that generates the set of valid subscripts for the specified dimension of a given array. Zero rows are returned for arrays that do not have the requested dimension, or for NULL arrays (but valid subscripts are returned for NULL array elements). Example:

```
-- unnest a 2D array
CREATE OR REPLACE FUNCTION unnest2(anyarray)
RETURNS SETOF anyelement AS $$
SELECT $1[i][j]
FROM generate_subscripts($1,1) g1(i),
```

```
generate_subscripts($1,2) g2(j);
$$ LANGUAGE sql IMMUTABLE;

SELECT * FROM unnest2(ARRAY[[1,2],[3,4]]);
unnest2
-----
 1
 2
 3
 4
(4 rows)
```

21.7.24 Geometric Functions and Operators

21.7.24.1 Geometric Operators

+

Description: The operator implements a panning function by adding the coordinates of the second point to each corresponding point of the first parameter.

Example:

```
SELECT box '((0,0),(1,1))' + point '(2.0,0)' AS RESULT;
result
-----
(3,1),(2,0)
(1 row)
```

-

Description: The operator implements a panning function by subtracting the coordinates of the second point from each corresponding point of the first parameter.

Example:

```
SELECT box '((0,0),(1,1))' - point '(2.0,0)' AS RESULT;
result
-----
(-1,1),(-2,0)
(1 row)
```

*

Description: Scaling out/rotation

Example:

```
SELECT box '((0,0),(1,1))' * point '(2.0,0)' AS RESULT;
result
-----
(2,2),(0,0)
(1 row)
```

/

Description: Scaling in/rotation

Example:

```
SELECT box '((0,0),(2,2))' / point '(2.0,0)' AS RESULT;  
result  
-----  
(1,1),(0,0)  
(1 row)
```

#

Description: Point or box of intersection

Example:

```
SELECT box '((1,-1),(-1,1))' # box '((1,1),(-1,-1))' AS RESULT;  
result  
-----  
(1,1),(-1,-1)  
(1 row)
```

#

Description: Number of paths or polygon vertexs

Example:

```
SELECT # path '((1,0),(0,1),(-1,0))' AS RESULT;  
result  
-----  
3  
(1 row)
```

@-@

Description: Length or circumference

Example:

```
SELECT @-@ path '((0,0),(1,0))' AS RESULT;  
result  
-----  
2  
(1 row)
```

@@

Description: Center of box

Example:

```
SELECT @@ circle '((0,0),10)' AS RESULT;  
result  
-----  
(0,0)  
(1 row)
```

##

Description: Closest point to first figure on second figure.

Example:

```
SELECT point '(0,0)' ## box '((2,0),(0,2))' AS RESULT;  
result  
-----
```

```
(0,0)
(1 row)
```

<->

Description: Distance between the two figures.

Example:

```
SELECT circle '((0,0),1)' <-> circle '((5,0),1)' AS RESULT;
result
-----
      3
(1 row)
```

&&

Description: Overlaps? (One point in common makes this true.)

Example:

```
SELECT box '((0,0),(1,1))' && box '((0,0),(2,2))' AS RESULT;
result
-----
      t
(1 row)
```

<<

Description: Is strictly left of (no common horizontal coordinate)?

Example:

```
SELECT circle '((0,0),1)' << circle '((5,0),1)' AS RESULT;
result
-----
      t
(1 row)
```

>>

Description: Is strictly right of (no common horizontal coordinate)?

Example:

```
SELECT circle '((5,0),1)' >> circle '((0,0),1)' AS RESULT;
result
-----
      t
(1 row)
```

&<

Description: Does not extend to the right of?

Example:

```
SELECT box '((0,0),(1,1))' &< box '((0,0),(2,2))' AS RESULT;
result
-----
      t
(1 row)
```

&>

Description: Does not extend to the left of?

Example:

```
SELECT box '((0,0),(3,3))' &> box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

<<|

Description: Is strictly below (no common horizontal coordinate)?

Example:

```
SELECT box '((0,0),(3,3))' <<| box '((3,4),(5,5))' AS RESULT;
result
-----
t
(1 row)
```

|>>

Description: Is strictly above (no common horizontal coordinate)?

Example:

```
SELECT box '((3,4),(5,5))' |>> box '((0,0),(3,3))' AS RESULT;
result
-----
t
(1 row)
```

&<|

Description: Does not extend above?

Example:

```
SELECT box '((0,0),(1,1))' &<| box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

|&>

Description: Does not extend below?

Example:

```
SELECT box '((0,0),(3,3))' |&> box '((0,0),(2,2))' AS RESULT;
result
-----
t
(1 row)
```

<^

Description: Is below (allows touching)?

Example:

```
SELECT box '((0,0),(-3,-3))' <^ box '((0,0),(2,2))' AS RESULT;  
result  
-----  
t  
(1 row)
```

>^

Description: Is above (allows touching)?

Example:

```
SELECT box '((0,0),(2,2))' >^ box '((0,0),(-3,-3))' AS RESULT;  
result  
-----  
t  
(1 row)
```

?#

Description: Intersect?

Example:

```
SELECT lseg '((-1,0),(1,0))' ?# box '((-2,-2),(2,2))' AS RESULT;  
result  
-----  
t  
(1 row)
```

?-

Description: Is horizontal?

Example:

```
SELECT ?- lseg '((-1,0),(1,0))' AS RESULT;  
result  
-----  
t  
(1 row)
```

?-

Description: Are horizontally aligned?

Example:

```
SELECT point '(1,0)' ?- point '(0,0)' AS RESULT;  
result  
-----  
t  
(1 row)
```

?|

Description: Is vertical?

Example:

```
SELECT ?| lseg '((-1,0),(1,0))' AS RESULT;  
result
```

```
-----  
f  
(1 row)
```

?|

Description: Are vertically aligned?

Example:

```
SELECT point '(0,1)' ?| point '(0,0)' AS RESULT;  
result  
-----  
t  
(1 row)
```

?-|

Description: Are perpendicular?

Example:

```
SELECT lseg '((0,0),(0,1))' ?-| lseg '((0,0),(1,0))' AS RESULT;  
result  
-----  
t  
(1 row)
```

?||

Description: Are parallel?

Example:

```
SELECT lseg '((-1,0),(1,0))' ?|| lseg '((-1,2),(1,2))' AS RESULT;  
result  
-----  
t  
(1 row)
```

@>

Description: Contains?

Example:

```
SELECT circle '((0,0),2)' @> point '(1,1)' AS RESULT;  
result  
-----  
t  
(1 row)
```

<@

Description: Contained in or on?

Example:

```
SELECT point '(1,1)' <@ circle '((0,0),2)' AS RESULT;  
result  
-----  
t  
(1 row)
```


~=

Description: Same as?

Example:

```
SELECT polygon '((0,0),(1,1))' ~= polygon '((1,1),(0,0))' AS RESULT;  
result  
-----  
t  
(1 row)
```

21.7.24.2 Geometric Functions

area(object)

Description: Area calculation

Return type: double precision

Example:

```
SELECT area(box '((0,0),(1,1))') AS RESULT;  
result  
-----  
1  
(1 row)
```

center(object)

Description: Figure center calculation

Return type: point

Example:

```
SELECT center(box '((0,0),(1,2))') AS RESULT;  
result  
-----  
(0.5,1)  
(1 row)
```

diameter(circle)

Description: Circle diameter calculation

Return type: double precision

Example:

```
SELECT diameter(circle '((0,0),2.0)') AS RESULT;  
result  
-----  
4  
(1 row)
```

height(box)

Description: Vertical size of box

Return type: double precision

Example:

```
SELECT height(box '((0,0),(1,1))') AS RESULT;  
result  
-----  
1  
(1 row)
```

isclosed(path)

Description: A closed path?

Return type: boolean

Example:

```
SELECT isclosed(path '((0,0),(1,1),(2,0))') AS RESULT;  
result  
-----  
t  
(1 row)
```

isopen(path)

Description: An open path?

Return type: boolean

Example:

```
SELECT isopen(path '[(0,0),(1,1),(2,0)]') AS RESULT;  
result  
-----  
t  
(1 row)
```

length(object)

Description: Length calculation

Return type: double precision

Example:

```
SELECT length(path '((-1,0),(1,0))') AS RESULT;  
result  
-----  
4  
(1 row)
```

npoints(path)

Description: Number of points in path

Return type: int

Example:

```
SELECT npoints(path '[(0,0),(1,1),(2,0)]') AS RESULT;  
result  
-----  
3  
(1 row)
```

npoints(polygon)

Description: Number of points in polygon

Return type: int

Example:

```
SELECT npoints(polygon '((1,1),(0,0))) AS RESULT;  
result  
-----  
2  
(1 row)
```

pclose(path)

Description: Converts path to closed.

Return type: path

Example:

```
SELECT pclose(path '[(0,0),(1,1),(2,0)]') AS RESULT;  
result  
-----  
((0,0),(1,1),(2,0))  
(1 row)
```

popen(path)

Description: Converts path to open.

Return type: path

Example:

```
SELECT popen(path '[(0,0),(1,1),(2,0)]') AS RESULT;  
result  
-----  
[(0,0),(1,1),(2,0)]  
(1 row)
```

radius(circle)

Description: Circle diameter calculation

Return type: double precision

Example:

```
SELECT radius(circle '((0,0),2.0)') AS RESULT;  
result  
-----  
2  
(1 row)
```

width(box)

Description: Horizontal size of box

Return type: double precision

Example:

```
SELECT width(box '((0,0),(1,1))') AS RESULT;  
result  
-----  
1  
(1 row)
```

21.7.24.3 Geometric Type Conversion Functions

box(circle)

Description: Circle to box

Return type: box

Example:

```
SELECT box(circle '((0,0),2.0)') AS RESULT;  
result  
-----  
(1.41421356237309,1.41421356237309),(-1.41421356237309,-1.41421356237309)  
(1 row)
```

box(point, point)

Description: Points to box

Return type: box

Example:

```
SELECT box(point '(0,0)', point '(1,1)') AS RESULT;  
result  
-----  
(1,1),(0,0)  
(1 row)
```

box(polygon)

Description: Polygon to box

Return type: box

Example:

```
SELECT box(polygon '((0,0),(1,1),(2,0))') AS RESULT;  
result  
-----  
(2,1),(0,0)  
(1 row)
```

circle(box)

Description: Box to circle

Return type: circle

Example:

```
SELECT circle(box '((0,0),(1,1))') AS RESULT;  
result  
-----  
<(0.5,0.5),0.707106781186548>  
(1 row)
```

circle(point, double precision)

Description: Center and radius to circle

Return type: circle

Example:

```
SELECT circle(point '(0,0)', 2.0) AS RESULT;  
result  
-----  
<(0,0),2>  
(1 row)
```

circle(polygon)

Description: Polygon to circle

Return type: circle

Example:

```
SELECT circle(polygon '((0,0),(1,1),(2,0))') AS RESULT;  
result  
-----  
<(1,0.3333333333333333),0.924950591148529>  
(1 row)
```

lseg(box)

Description: Box diagonal to line segment

Return type: lseg

Example:

```
SELECT lseg(box '((-1,0),(1,0))') AS RESULT;  
result  
-----  
[(1,0),(-1,0)]  
(1 row)
```

lseg(point, point)

Description: Points to line segment

Return type: lseg

Example:

```
SELECT lseg(point '(-1,0)', point '(1,0)') AS RESULT;  
result  
-----  
[(-1,0),(1,0)]  
(1 row)
```

path(polygon)

Description: Polygon to path

Return type: path

Example:

```
SELECT path(polygon '((0,0),(1,1),(2,0)))' AS RESULT;  
result  
-----  
((0,0),(1,1),(2,0))  
(1 row)
```

point(double precision, double precision)

Description: Points

Return type: point

Example:

```
SELECT point(23.4, -44.5) AS RESULT;  
result  
-----  
(23.4,-44.5)  
(1 row)
```

point(box)

Description: Center of box

Return type: point

Example:

```
SELECT point(box '((-1,0),(1,0))') AS RESULT;  
result  
-----  
(0,0)  
(1 row)
```

point(circle)

Description: Center of circle

Return type: point

Example:

```
SELECT point(circle '((0,0),2.0)') AS RESULT;  
result  
-----  
(0,0)  
(1 row)
```

point(lseg)

Description: Center of line segment

Return type: point

Example:

```
SELECT point(lseg '((-1,0),(1,0))') AS RESULT;  
result  
-----  
(0,0)  
(1 row)
```

point(polygon)

Description: Center of polygon

Return type: point

Example:

```
SELECT point(polygon '((0,0),(1,1),(2,0))') AS RESULT;  
result  
-----  
(1,0.3333333333333333)  
(1 row)
```

polygon(box)

Description: Box to 4-point polygon

Return type: polygon

Example:

```
SELECT polygon(box '((0,0),(1,1))') AS RESULT;  
result  
-----  
((0,0),(0,1),(1,1),(1,0))  
(1 row)
```

polygon(circle)

Description: Circle to 12-point polygon

Return type: polygon

Example:

```
SELECT polygon(circle '((0,0),2.0)') AS RESULT;  
result  
-----  
-----  
-----  
((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),(1,1.73205080756888),  
(1.73205080756888,1),(2.244929359829471e-16),(1.73205080756888,-0.9999999999999999),  
(1,-1.73205080756888),(3.67394039744206e-16,-2),(-0.9999999999999999,-1.73205080756888),  
(-1.73205080756888,-1))  
(1 row)
```

polygon(npts, circle)

Description: Circle to **npts**-point polygon

Return type: polygon

Example:

```
SELECT polygon(12, circle '((0,0),2.0)') AS RESULT;  
result  
-----  
-----  
-----  
((-2,0),(-1.73205080756888,1),(-1,1.73205080756888),(-1.22464679914735e-16,2),(1,1.73205080756888),
```

```
(1.73205080756888,1),(2,2.44929359829471e-16),(1.73205080756888,-0.999999999999999),  
(1,-1.73205080756888),(3.67394039744206e-16,-2),(-0.999999999999999,-1.73205080756888),  
(-1.73205080756888,-1))  
(1 row)
```

polygon(path)

Description: Path to polygon

Return type: polygon

Example:

```
SELECT polygon(path '((0,0),(1,1),(2,0))') AS RESULT;  
result  
-----  
((0,0),(1,1),(2,0))  
(1 row)
```

21.7.25 Network Address Functions and Operators

21.7.25.1 cidr and inet Operators

The operators <<, <=, >>, and >= test for subnet inclusion. They consider only the network parts of the two addresses (ignoring any host part) and determine whether one network is identical to or a subnet of the other.

<

Description: Is less than

Example:

```
SELECT inet '192.168.1.5' < inet '192.168.1.6' AS RESULT;  
result  
-----  
t  
(1 row)
```

<=

Description: Is less than or equals

Example:

```
SELECT inet '192.168.1.5' <= inet '192.168.1.5' AS RESULT;  
result  
-----  
t  
(1 row)
```

=

Description: Equals

Example:

```
SELECT inet '192.168.1.5' = inet '192.168.1.5' AS RESULT;  
result  
-----
```



```
t  
(1 row)
```

>=

Description: Is greater than or equals

Example:

```
SELECT inet '192.168.1.5' >= inet '192.168.1.5' AS RESULT;  
result  
-----  
t  
(1 row)
```

>

Description: Is greater than

Example:

```
SELECT inet '192.168.1.5' > inet '192.168.1.4' AS RESULT;  
result  
-----  
t  
(1 row)
```

<>

Description: Does not equal to

Example:

```
SELECT inet '192.168.1.5' <> inet '192.168.1.4' AS RESULT;  
result  
-----  
t  
(1 row)
```

<<

Description: Is contained in

Example:

```
SELECT inet '192.168.1.5' << inet '192.168.1/24' AS RESULT;  
result  
-----  
t  
(1 row)
```

<<=

Description: Is contained in or equals

Example:

```
SELECT inet '192.168.1/24' <<= inet '192.168.1/24' AS RESULT;  
result  
-----  
t  
(1 row)
```

>>

Description: Contains

Example:

```
SELECT inet '192.168.1/24' >> inet '192.168.1.5' AS RESULT;  
result  
-----  
t  
(1 row)
```

>>=

Description: Contains or equals

Example:

```
SELECT inet '192.168.1/24' >>= inet '192.168.1/24' AS RESULT;  
result  
-----  
t  
(1 row)
```

~

Description: Bitwise NOT

Example:

```
SELECT ~ inet '192.168.1.6' AS RESULT;  
result  
-----  
63.87.254.249  
(1 row)
```

&

Description: The AND operation is performed on each bit of the two network addresses.

Example:

```
SELECT inet '192.168.1.6' & inet '10.0.0.0' AS RESULT;  
result  
-----  
0.0.0.0  
(1 row)
```

|

Description: The OR operation is performed on each bit of the two network addresses.

Example:

```
SELECT inet '192.168.1.6' | inet '10.0.0.0' AS RESULT;  
result  
-----  
202.168.1.6  
(1 row)
```

+

Description: Addition

Example:

```
SELECT inet '192.168.1.6' + 25 AS RESULT;  
result  
-----  
192.168.1.31  
(1 row)
```

-

Description: Subtraction

Example:

```
SELECT inet '192.168.1.43' - 36 AS RESULT;  
result  
-----  
192.168.1.7  
(1 row)  
SELECT inet '192.168.1.43' - inet '192.168.1.19' AS RESULT;  
result  
-----  
24  
(1 row)
```

21.7.25.2 Network Address Functions

The **abbrev**, **host**, and **text** functions are primarily intended to offer alternative display formats.

Any **cidr** value can be cast to **inet** implicitly or explicitly; therefore, the functions shown above as operating on **inet** also work on **cidr** values. An **inet** value can be cast to **cidr**. After the conversion, any bits to the right of the subnet mask are silently zeroed to create a valid **cidr** value. In addition, you can cast a text string to **inet** or **cidr** using normal casting syntax. For example, **inet(expression)** or **colname::cidr**.

abbrev(inet)

Description: Abbreviated display format as text

Return type: text

Example:

```
SELECT abbrev(inet '10.1.0.0/16') AS RESULT;  
result  
-----  
10.1.0.0/16  
(1 row)
```

abbrev(cidr)

Description: Abbreviated display format as text

Return type: text

Example:

```
SELECT abbrev(cidr '10.1.0.0/16') AS RESULT;  
result  
-----  
10.1/16  
(1 row)
```

broadcast(inet)

Description: Broadcast address for network

Return type: inet

Example:

```
SELECT broadcast('192.168.1.5/24') AS RESULT;  
result  
-----  
192.168.1.255/24  
(1 row)
```

family(inet)

Description: Extracts family of address; **4** for IPv4, **6** for IPv6

Return type: int

Example:

```
SELECT family('::1') AS RESULT;  
result  
-----  
6  
(1 row)
```

host(inet)

Description: Extracts IP address as text.

Return type: text

Example:

```
SELECT host('192.168.1.5/24') AS RESULT;  
result  
-----  
192.168.1.5  
(1 row)
```

hostmask(inet)

Description: Constructs host mask for network.

Return type: inet

Example:

```
SELECT hostmask('192.168.23.20/30') AS RESULT;  
result  
-----  
0.0.0.3  
(1 row)
```

masklen(inet)

Description: Extracts subnet mask length.

Return type: int

Example:

```
SELECT masklen('192.168.1.5/24') AS RESULT;  
result  
-----  
24  
(1 row)
```

netmask(inet)

Description: Constructs a subnet mask for the network.

Return type: inet

Example:

```
SELECT netmask('192.168.1.5/24') AS RESULT;  
result  
-----  
255.255.255.0  
(1 row)
```

network(inet)

Description: Extracts network part of address.

Return type: cidr

Example:

```
SELECT network('192.168.1.5/24') AS RESULT;  
result  
-----  
192.168.1.0/24  
(1 row)
```

set_masklen(inet, int)

Description: Sets subnet mask length for **inet** value.

Return type: inet

Example:

```
SELECT set_masklen('192.168.1.5/24', 16) AS RESULT;  
result  
-----  
192.168.1.5/16  
(1 row)
```

set_masklen(cidr, int)

Description: Sets subnet mask length for **cidr** value.

Return type: cidr

Example:

```
SELECT set_masklen('192.168.1.0/24'::cidr, 16) AS RESULT;  
result  
-----  
192.168.0.0/16  
(1 row)
```

text(inet)

Description: Extracts IP address and subnet mask length as text.

Return type: text

Example:

```
SELECT text(inet '192.168.1.5') AS RESULT;  
result  
-----  
192.168.1.5/32  
(1 row)
```

trunc(macaddr)

The function **trunc(macaddr)** returns a MAC address with the last 3 bytes set to zero.

Description: Sets last 3 bytes to zero.

Return type: macaddr

Example:

```
SELECT trunc(macaddr '12:34:56:78:90:ab') AS RESULT;  
result  
-----  
12:34:56:00:00:00  
(1 row)
```

The **macaddr** type also supports the standard relational operators (such as > and <=) for lexicographical ordering, and the bitwise arithmetic operators (~, & and |) for NOT, AND and OR.

21.7.26 System Information Functions

21.7.26.1 Session Information Functions

current_catalog

Description: name of the current database (called **catalog** in the SQL standard), same as **current_database()**.

Return type: name

Example:

```
SELECT current_catalog;  
current_database  
-----  
gaussdb  
(1 row)
```

current_database()

Description: Name of the current database

Return type: name

Example:

```
SELECT current_database();
current_database
-----
gaussdb
(1 row)
```

pg_client_encoding()

Description: Obtains the character encoding mode of the current client. When characters are displayed incorrectly, you can check the client encoding to quickly locate the fault.

Return type: name

Example:

```
SELECT pg_client_encoding();
pg_client_encoding
-----
UTF8
(1 row)
```

current_query()

Description: Text of the currently executing query, as submitted by the client (might contain more than one statement)

Return type: text

Example:

```
SELECT current_query();
current_query
-----
SELECT current_query();
(1 row)
```

current_schema[()]

Description: **current_schema** returns the first valid schema name in the search path. (If the search path is empty or contains no valid schema name, **NULL** is returned.) This is the schema that will be used for any tables or other named objects that are created without specifying a target schema.

Return type: name

Example:

```
SELECT current_schema();
current_schema
-----
public
(1 row)
```

current_schemas(boolean)

Description: **current_schemas(boolean)** returns an array of the names of all schemas presently in the search path. The Boolean option determines whether implicitly included system schemas such as **pg_catalog** are included in the returned search path.

Return type: name[]

Example:

```
SELECT current_schemas(true);
 current_schemas
-----
{pg_catalog,public}
(1 row)
```

NOTE

The search path can be altered at run time. The command is:

```
SET search_path TO schema [, schema, ...]
```

current_user

Description: Username of current execution context. **current_user** is the identifier of the user whose permission needs to be checked. It is usually used to represent a session user, but this setting can be changed according to [SET ROLE](#). It also changes during the execution of functions with the attribute **SECURITY DEFINER**.

Return type: name

Example:

```
SELECT current_user;
 current_user
-----
dbadmin
(1 row)
```

inet_client_addr()

Description: Displays the IP address of the currently connected client.

NOTE

- It is available only in remote connection mode.
- If the database is connected to the local PC, the value is empty.

Return type: inet

Example:

```
SELECT inet_client_addr();
 inet_client_addr
-----
10.10.0.50
(1 row)
```

inet_client_port()

Description: Displays the port number of the currently connected client.

 NOTE

It is available only in remote connection mode.

Return type: integer

Example:

```
SELECT inet_client_port();
inet_client_port
-----
33143
(1 row)
```

inet_server_addr()

Description: Displays the IP address of the current server.

 NOTE

- It is available only in remote connection mode.
- If the database is connected to the local PC, the value is empty.

Return type: inet

Example:

```
SELECT inet_server_addr();
inet_server_addr
-----
10.10.0.13
(1 row)
```

inet_server_port()

Description: Displays the port of the current server. All these functions return NULL if the current connection is via a Unix-domain socket.

 NOTE

It is available only in remote connection mode.

Return type: integer

Example:

```
SELECT inet_server_port();
inet_server_port
-----
8000
(1 row)
```

pg_backend_pid()

Description: Process ID of the server process attached to the current session

Return type: integer

Example:

```
SELECT pg_backend_pid();
pg_backend_pid
-----
```

```
140229352617744  
(1 row)
```

pg_conf_load_time()

Description: Configures load time. **pg_conf_load_time** returns the timestamp with time zone when the server configuration files were last loaded.

Return type: timestamp with time zone

Example:

```
SELECT pg_conf_load_time();  
       pg_conf_load_time  
-----  
2017-09-01 16:05:23.89868+08  
(1 row)
```

pg_my_temp_schema()

Description: **pg_my_temp_schema** returns the OID of the current session's temporary schema, or zero if it has none (because it has not created any temporary tables). **pg_is_other_temp_schema** returns true if the given OID is the OID of another session's temporary schema.

Return type: OID

Example:

```
SELECT pg_my_temp_schema();  
       pg_my_temp_schema  
-----  
0  
(1 row)
```

pg_is_other_temp_schema(oid)

Description: Whether the schema is the temporary schema of another session.

Return type: boolean

Example:

```
SELECT pg_is_other_temp_schema(25356);  
       pg_is_other_temp_schema  
-----  
f  
(1 row)
```

pg_postmaster_start_time()

Description: Specifies the start time of the database instance.

pg_postmaster_start_time returns the timestamp (with time zone) when the database instance was started.

Return type: timestamp with time zone

Example:

```
SELECT pg_postmaster_start_time();  
       pg_postmaster_start_time  
-----
```

```
2017-08-30 16:02:54.99854+08
(1 row)
```

pg_trigger_depth()

Description: Current nesting level of triggers

Return type: integer

Example:

```
SELECT pg_trigger_depth();
pg_trigger_depth
-----
0
(1 row)
```

pgxc_version()

Description: Postgres-XC version information

Return type: text

Example:

```
SELECT pgxc_version();
pgxc_version
-----
Postgres-XC 1.1 on x86_64-unknown-linux-gnu, based on PostgreSQL 9.2.4, compiled by g++ (GCC) 5.4.0,
64-bit
(1 row)
```

session_user

Description: Session user name **session_user** is usually the user who initiated the current database connection, but administrators can change this setting with [SET SESSION AUTHORIZATION](#).

Return type: name

Example:

```
SELECT session_user;
session_user
-----
dbadmin
(1 row)
```

user

Description: Is equivalent to **current_user**.

Return type: name

Example:

```
SELECT user;
current_user
-----
dbadmin
(1 row)
```

version()

Description: version information. **version** returns a string describing a server's version.

Return type: text

Example:

```
SELECT version();
```

| version |
|---|
| PostgreSQL 9.2.4 gsqr ((GaussDB 8.2.1 build 39137c2d) compiled at 2022-09-23 15:43:11 commit 3629 last
mr 5138 release) on x86_64-unknown-linux-gnu, compiled by g++ (GCC) 5.4.0, 64-bit |

21.7.26.2 Access Privilege Inquiry Functions

has_any_column_privilege(user, table, privilege)

Description: Queries whether a specified user has permission for any column of table.

Return type: boolean

has_any_column_privilege(table, privilege)

Description: Queries whether the current user has permission for any column of table.

Return type: boolean

has_any_column_privilege checks whether a user can access any column of a table in a particular way. Its parameter possibilities are analogous to **has_table_privilege**, except that the desired access permission type must be some combination of SELECT, INSERT, UPDATE, or REFERENCES.

NOTE

Note that having any of these permissions at the table level implicitly grants it for each column of the table, so **has_any_column_privilege** will always return **true** if **has_table_privilege** does for the same parameters. But **has_any_column_privilege** also succeeds if there is a column-level grant of the permission for at least one column.

has_column_privilege(user, table, column, privilege)

Description: Queries whether a specified user has permission for column.

Return type: boolean

has_column_privilege(table, column, privilege)

Description: Queries whether the current user has permission for column.

Return type: boolean

has_column_privilege checks whether a user can access a column in a particular way. Its argument possibilities are analogous to **has_table_privilege**, with the

addition that the column can be specified either by name or attribute number. The desired access permission type must evaluate to some combination of **SELECT**, **INSERT**, **UPDATE**, or **REFERENCES**.

 NOTE

Note that having any of these permissions at the table level implicitly grants it for each column of the table.

has_database_privilege(user, database, privilege)

Description: Queries whether a specified user has permission for database.

Return type: boolean

has_database_privilege(database, privilege)

Description: Queries whether the current user has permission for database.

Return type: boolean

Note: **has_database_privilege** checks whether a user can access a database in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission type must evaluate to some combination of **CREATE**, **CONNECT**, **TEMPORARY**, or **TEMP** (which is equivalent to **TEMPORARY**).

has_foreign_data_wrapper_privilege(user, fdw, privilege)

Description: Queries whether a specified user has permission for foreign-data wrapper.

The **fdw** parameter indicates the name or ID of the foreign data wrapper.

Return type: boolean

has_foreign_data_wrapper_privilege(fdw, privilege)

Description: Queries whether the current user has permission for foreign-data wrapper.

Return type: boolean

Note: **has_foreign_data_wrapper_privilege** checks whether a user can access a foreign-data wrapper in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission type must evaluate to **USAGE**.

has_function_privilege(user, function, privilege)

Description: Queries whether a specified user has permission for function.

Return type: boolean

has_function_privilege(function, privilege)

Description: Queries whether the current user has permission for function.

Return type: boolean

Note: **has_function_privilege** checks whether a user can access a function in a particular way. Its argument possibilities are analogous to **has_table_privilege**. When a function is specified by a text string rather than by OID, the allowed input is the same as that for the **regprocedure** data type (see [Object Identifier Types](#)). The desired access permission type must evaluate to **EXECUTE**.

has_language_privilege(user, language, privilege)

Description: Queries whether a specified user has permission for language.

Return type: boolean

has_language_privilege(language, privilege)

Description: Queries whether the current user has permission for language.

Return type: boolean

Note: **has_language_privilege** checks whether a user can access a procedural language in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission type must evaluate to **USAGE**.

has_schema_privilege(user, schema, privilege)

Description: Queries whether a specified user has permission for schema.

Return type: boolean

has_schema_privilege(schema, privilege)

Description: Queries whether the current user has permission for schema.

Return type: boolean

Note: **has_schema_privilege** checks whether a user can access a schema in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission type must evaluate to some combination of **CREATE** or **USAGE**.

has_server_privilege(user, server, privilege)

Description: Queries whether a specified user has permission for foreign server.

Return type: boolean

has_server_privilege(server, privilege)

Description: Queries whether the current user has permission for foreign server.

Return type: boolean

Note: **has_server_privilege** checks whether a user can access a foreign server in a particular way. Its argument possibilities are analogous to **has_table_privilege**. The desired access permission type must evaluate to **USAGE**.

has_table_privilege(user, table, privilege)

Description: Queries whether a specified user has permission for table.

Return type: boolean

has_table_privilege(table, privilege)

Description: Queries whether the current user has permission for table.

Return type: boolean

has_table_privilege checks whether a user can access a table in a particular way. The user can be specified by name, by OID (**pg_authid.oid**), **public** to indicate the PUBLIC pseudo-role, or if the argument is omitted **current_user** is assumed. The table can be specified by name or by OID. When specifying by name, the name can be schema-qualified if necessary. The desired access permission type is specified by a text string, which must be one of the values **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **TRUNCATE**, **REFERENCES**, or **TRIGGER**. Optionally, **WITH GRANT OPTION** can be added to a permission type to test whether the permission is held with grant option. Also, multiple permission types can be listed separated by commas, in which case the result will be **true** if any of the listed permissions is held.

Example:

```
SELECT has_table_privilege('tpcds.web_site', 'select');
has_table_privilege
-----
t
(1 row)

SELECT has_table_privilege('dbadmin', 'tpcds.web_site', 'select,INSERT WITH GRANT OPTION ');
has_table_privilege
-----
t
(1 row)
```

pg_has_role(user, role, privilege)

Description: Queries whether a specified user has permission for role.

Return type: boolean

pg_has_role(role, privilege)

Description: Specifies whether the current user has permission for role.

Return type: boolean

Note: **pg_has_role** checks whether a user can access a role in a particular way. Its argument possibilities are analogous to **has_table_privilege**, except that **public** is not allowed as a user name. The desired access permission type must evaluate to some combination of **MEMBER** or **USAGE**. **MEMBER** denotes direct or indirect membership in the role (that is, the right to do **SET ROLE**), while **USAGE** denotes the permissions of the role are available without doing **SET ROLE**.

21.7.26.3 Schema Visibility Inquiry Functions

Schema visibility inquiry functions perform visibility checks on database objects. For functions and operators, an object in the search path is visible if there is no object of the same name and argument data type(s) earlier in the path. For operator classes, both name and associated index access method are considered.

All these functions require OIDs to identify the objects to be checked. If you want to test an object by name, it is convenient to use the OID alias types (**regclass**, **regtype**, **regprocedure**, **regoperator**, **regconfig**, or **regdictionary**).

For example, a table is said to be visible if its containing schema is in the search path and no table of the same name appears earlier in the search path. This is equivalent to the statement that the table can be referenced by name without explicit schema qualification. For example, to list the names of all visible tables:

```
SELECT relname FROM pg_class WHERE pg_table_is_visible(oid);
```

pg_collation_is_visible(collation_oid)

Description: Queries whether the collation is visible in search path.

Return type: boolean

pg_conversion_is_visible(conversion_oid)

Description: Queries whether the conversion is visible in search path.

Return type: boolean

pg_function_is_visible(function_oid)

Description: Queries whether the function is visible in search path.

Return type: boolean

pg_opclass_is_visible(opclass_oid)

Description: Queries whether the operator class is visible in search path.

Return type: boolean

pg_operator_is_visible(operator_oid)

Description: Queries whether the operator is visible in search path.

Return type: boolean

pg_opfamily_is_visible(opclass_oid)

Description: Queries whether the operator family is visible in search path.

Return type: boolean

pg_table_is_visible(table_oid)

Description: Queries whether the table is visible in search path.

Return type: boolean

pg_ts_config_is_visible(config_oid)

Description: Queries whether the text search configuration is visible in search path.

Return type: boolean

pg_ts_dict_is_visible(dict_oid)

Description: Queries whether the text search dictionary is visible in search path.

Return type: boolean

pg_ts_parser_is_visible(parser_oid)

Description: Queries whether the text search parser is visible in search path.

Return type: boolean

pg_ts_template_is_visible(template_oid)

Description: Queries whether the text search template is visible in search path.

Return type: boolean

pg_type_is_visible(type_oid)

Description: Queries whether the type (or domain) is visible in search path.

Return type: boolean

21.7.26.4 System Catalog Information Functions

format_type(type_oid, typemod)

Description: Gets SQL name of a data type.

Return type: text

Note:

format_type returns the SQL name of a data type that is identified by its type OID and possibly a type modifier. Pass NULL for the type modifier if no specific modifier is known. Certain type modifiers are passed for data types with length limitations. The SQL name returned from **format_type** contains the length of the data type, which can be calculated by taking `sizeof(int32)` from actual storage length [actual storage len - `sizeof(int32)`] in the unit of bytes. 32-bit space is required to store the customized length set by users. So the actual storage length contains 4 bytes more than the customized length. In the following example, the SQL name returned from **format_type** is `character varying(6)`, indicating the length of varchar type is 6 bytes. So the actual storage length of varchar type is 10 bytes.

```
SELECT format_type((SELECT oid FROM pg_type WHERE typname='varchar'), 10);
format_type
```

```
-----  
character varying(6)  
(1 row)
```

pg_check_authid(role_oid)

Description: Check whether a role name with given OID exists.

Return type: Boolean

pg_describe_object(catalog_id, object_id, object_sub_id)

Description: Gets description of a database object.

Return type: text

Note: **pg_describe_object** returns a description of a database object specified by catalog OID, object OID and a (possibly zero) sub-object ID. This is useful to determine the identity of an object as stored in the **pg_depend** catalog.

pg_get_constraintdef(constraint_oid)

Description: Gets definition of a constraint.

Return type: text

pg_get_constraintdef(constraint_oid, pretty_bool)

Description: Gets definition of a constraint.

Return type: text

Note: **pg_get_constraintdef** and **pg_get_indexdef** respectively reconstruct the creating command for a constraint and an index.

pg_get_expr(pg_node_tree, relation_oid)

Description: Decompiles internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.

Return type: text

pg_get_expr(pg_node_tree, relation_oid, pretty_bool)

Description: Decompiles internal form of an expression, assuming that any Vars in it refer to the relationship indicated by the second parameter.

Return type: text

Note: **pg_get_expr** decompiles the internal form of an individual expression, such as the default value for a column. It can be useful when examining the contents of system catalogs. If the expression might contain Vars, specify the OID of the relationship they refer to as the second parameter; if no Vars are expected, zero is sufficient.

pg_get_indexdef(index_oid)

Description: Gets **CREATE INDEX** command for index.

Return type: text

index_oid indicates the index OID, which can be queried in the **PG_STATIO_ALL_INDEXES** system view.

Example: Query the OID and CREATE INDEX command of **index ds_ship_mode_t1_index1**.

```
SELECT indexrelid FROM PG_STATIO_ALL_INDEXES WHERE indexrelname = 'ds_ship_mode_t1_index1';
indexrelid
-----
136035
(1 row)
SELECT * FROM pg_get_indexdef(136035);
pg_get_indexdef
-----
CREATE INDEX ds_ship_mode_t1_index1 ON tpceds.ship_mode_t1 USING psort (sm_ship_mode_sk)
TABLESPACE pg_default
(1 row)
```

pg_get_indexdef(index_oid, column_no, pretty_bool)

Description: Gets **CREATE INDEX** command for index, or definition of just one index column when **column_no** is not zero.

Return type: text

```
SELECT * FROM pg_get_indexdef(136035,0,false);
pg_get_indexdef
-----
CREATE INDEX ds_ship_mode_t1_index1 ON tpceds.ship_mode_t1 USING psort (sm_ship_mode_sk)
TABLESPACE pg_default
(1 row)
SELECT * FROM pg_get_indexdef(136035,1,false);
pg_get_indexdef
-----
sm_ship_mode_sk
(1 row)
```

pg_get_keywords()

Description: Gets list of SQL keywords and their categories.

Return type: SETOF record

Note: **pg_get_keywords** returns a set of records describing the SQL keywords recognized by the server. The **word** column contains the keyword. The **catcode** column contains a category code: **U** for unreserved, **C** for column name, **T** for type or function name, or **R** for reserved. The **catdesc** column contains a possibly-localized string describing the category.

pg_get_ruledef(rule_oid)

Description: Gets **CREATE RULE** command for a rule.

Return type: text

pg_get_ruledef(rule_oid, pretty_bool)

Description: Gets **CREATE RULE** command for a rule.

Return type: text

pg_get_userbyid(role_oid)

Description: Gets role name with given OID.

Return type: name

Note: **pg_get_userbyid** extracts a role's name given its OID.

pg_get_viewdef(viewname text [, pretty bool [, fullflag bool]])

Description: gets underlying **SELECT** command for views.

Return type: text

Note:

- **pg_get_viewdef** reconstructs the **SELECT** query that defines a view. If the value of **pretty bool** is set to **true**, the display format is suitable for printing and more readable. The default value of **pretty bool** is **false**, and the display format is not readable. Use the default format for dump purposes whenever possible. The **pretty bool** parameter can be applied only to valid views.
- When **fullflag bool** is set to **true**, the complete definition of the view is displayed. The default value is **false**.

pg_get_viewdef(viewoid oid [, pretty bool [, fullflag bool]])

Description: gets underlying **SELECT** command for views.

Return type: text

pg_get_viewdef(view_oid, wrap_column_int)

Description: Gets underlying **SELECT** command for view, wrapping lines with columns as specified, printing is implied.

Return type: text

pg_get_tabledef(table_oid)

Description: Obtains the table definition based on the table OID.

Return type: text

Example: Obtain the OID of the table **customer_t2** from the system catalog **pg_class**, and then use this function to query the definition of **customer_t2** to obtain the table columns, storage mode (row-store or column-store), and table distribution mode configured for **customer_t2** when it is created.

```
select oid from pg_class where relname ='customer_t2';
oid
-----
17353
```

```
(1 row)

select * from pg_get_tabledef(17353);
      pg_get_tabledef
-----
SET search_path = dbadmin;          +
CREATE TABLE customer_t2 (        +
  state_id character(2),             +
  state_name character varying(40), +
  area_id numeric                    +
)                                    +
WITH (orientation=column, compression=low)+
DISTRIBUTE BY HASH(state_id)        +
TO GROUP group_version1;
(1 row)
```

pg_get_tabledef(table_name)

Description: Obtains the table definition based on the table name.

Return type: text

Remarks: **pg_get_tabledef** reconstructs the **CREATE** statement of the table definition, including the table definition, index information, and comments. Users need to create the dependent objects of the table, such as groups, schemas, tablespaces, and servers. The table definition does not include the statements for creating these dependent objects.

pg_get_tabledef(table_name/table_oid, forCreate boolean)

Description: Obtains the table definition based on the table name or OID. It is supported only by clusters of version 9.1.0.100 or later.

Return type: text

- If the **forCreate** parameter is **true**, the table definition obtained will exclude the **TO GROUP** clause.
- If the **forCreate** parameter is **false**, the table definition obtained will include the **TO GROUP** clause.

pg_get_tabledef(table_name/table_oid, forCreate boolean, withColComm boolean)

Description: Obtains the table definition based on the table name or OID. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: text

- If **withColComment** is **true**, the comment for a column field is displayed next to it,
- If **withColComment** is **false**, the comment for a column is shown at the end of the table definition.

Example:

```
SELECT pg_get_tabledef('person',false,true);
      pg_get_tabledef
-----
SET search_path = public;          +
CREATE TABLE person (             +
```

```
id integer COMMENT 'Student ID',      +
name character varying(25) COMMENT 'Name', +
address text COMMENT 'Address'      +
)                                     +
WITH (orientation=row, compression=no) +
DISTRIBUTE BY ROUNDROBIN             +
TO GROUP group_version1;
(1 row)

SELECT pg_get_tabledef('person',false,false);
pg_get_tabledef
-----
SET search_path = public;      +
CREATE TABLE person (        +
id integer,                    +
name character varying(25),    +
address text                   +
)                               +
WITH (orientation=row, compression=no) +
DISTRIBUTE BY ROUNDROBIN      +
TO GROUP group_version1;      +
COMMENT ON COLUMN person.id IS 'Student ID'; +
COMMENT ON COLUMN person.name IS 'Name'; +
COMMENT ON COLUMN person.address IS 'Address';
(1 row)
```

pg_get_stat_expressions(oid)

Description: Obtains the expressions in the extended statistics objects. Only clusters of version 8.2.1.200 or later support this function.

Return type: text[]

pg_options_to_table(reloptions)

Description: Gets the set of storage option name/value pairs.

Return type: SETOF record

Note: **pg_options_to_table** returns the set of storage option name/value pairs (**option_name/option_value**) when passing **pg_class.reloptions** or **pg_attribute.attoptions**.

Example:

```
CREATE TABLE customer_test
(
state_ID CHAR(2),
state_NAME VARCHAR2(40),
area_ID NUMBER
)
WITH (ORIENTATION = COLUMN, COMPRESSION=middle);
SELECT pg_options_to_table(reloptions) FROM pg_class WHERE relname='customer_test';
pg_options_to_table
-----
(orientation,column)
(compression,middle)
(bucketnums,16384)
(colversion,2.0)
(enable_delta,false)
(5 rows)
```

pg_typeof(any)

Description: Gets the data type of any value.

Return type: regtype

Note:

pg_typeof returns the OID of the data type of the value that is passed to it. This can be helpful for troubleshooting or dynamically constructing SQL queries. The function is declared as returning **regtype**, which is an OID alias type (see [Object Identifier Types](#)). This means that it is the same as an OID for comparison purposes but displays as a type name.

Example:

```
SELECT pg_typeof(33);
pg_typeof
-----
integer
(1 row)

SELECT typen FROM pg_type WHERE oid = pg_typeof(33);
typen
-----
4
(1 row)
```

collation for (any)

Description: Gets the collation of the parameter.

Return type: text

Note:

The expression **collation for** returns the collation of the value that is passed to it. Example:

```
SELECT collation for (description) FROM pg_description LIMIT 1;
pg_collation_for
-----
"default"
(1 row)
```

The value might be quoted and schema-qualified. If no collation is derived for the argument expression, then a null value is returned. If the parameter is not of a collatable data type, then an error is thrown.

getdistributekey(table_name)

Description: Gets a distribution column for a hash table.

Return type: text

Example:

```
SELECT getdistributekey('item');
getdistributekey
-----
i_item_sk
(1 row)
```

21.7.26.5 System Function Checking Function

pv_builtin_functions()

Description: Queries information about system functions.

Return type: record

pg_get_functiondef(func_oid)

Description: Gets definition of a function.

Return type: text

func_oid is the OID of the function, which can be queried in the **PG_PROC** system catalog.

Example: Query the OID and definition of the **justify_days** function.

```
SELECT oid FROM pg_proc WHERE proname = 'justify_days';
 oid
-----
 1295
(1 row)

SELECT * FROM pg_get_functiondef(1295);
headerlines | definition
-----+-----
4 | CREATE OR REPLACE FUNCTION pg_catalog.justify_days(interval)+
  | RETURNS interval +
  | LANGUAGE internal +
  | IMMUTABLE STRICT NOT FENCED NOT SHIPPABLE +
  | AS $function$interval_justify_days$function$ +
(1 row)
```

Note: The query result returned by **pg_get_functiondef** is in the original text format of the stored procedure. The escape character (\) is used to facilitate subsequent parsing.

pg_get_function_arguments(func_oid)

Description: Gets argument list of function's definition (with default values).

Return type: text

Note: **pg_get_function_arguments** returns the argument list of a function, in the form it would need to appear in within **CREATE FUNCTION**.

pg_get_function_identity_arguments(func_oid)

Description: Gets argument list to identify a function (without default values).

Return type: text

Note: **pg_get_function_identity_arguments** returns the argument list necessary to identify a function, in the form it would need to appear in within **ALTER FUNCTION**. This form omits default values.

pg_get_function_result(func_oid)

Description: Gets **RETURNS** clause for function.

Return type: text

Note: **pg_get_function_result** returns the appropriate **RETURNS** clause for the function.

21.7.26.6 Comment Checking Functions

col_description(table_oid, column_number)

Description: Gets comment for a table column.

Return type: text

Note: **col_description** returns the comment for a table column, which is specified by the OID of its table and its column number.

Example: Query the **pg_class** system catalog to obtain the table OID, and query the INFORMATION_SCHEMA.COLUMNS system view to obtain **column_number**.

```
SELECT COLUMN_NAME,ORDINAL_POSITION FROM INFORMATION_SCHEMA.COLUMNS WHERE  
TABLE_NAME = 't' AND COLUMN_NAME = 'a';
```

```
column_name | ordinal_position
```

```
-----+-----  
a          |          1  
(1 row)
```

```
SELECT oid FROM pg_class WHERE relname='t';
```

```
oid
```

```
-----  
44020  
(1 row)
```

```
SELECT col_description(44020,1);  
col_description
```

```
-----  
This is a test table.  
(1 row)
```

obj_description(object_oid, catalog_name)

Description: Gets comment for a database object.

Return type: text

Note: The two-parameter form of **obj_description** returns the comment for a database object specified by its OID and the name of the containing system catalog. For example, **obj_description(123456,'pg_class')** would retrieve the comment for the table with OID 123456. The one-parameter form of **obj_description** requires only the object OID.

obj_description cannot be used for table columns since columns do not have OIDs of their own.

obj_description(object_oid)

Description: Gets comment for a database object.

Return type: text

shobj_description(object_oid, catalog_name)

Description: Gets comment for a shared database object.

Return type: text

Note: The functions of **shobj_description** and **obj_description** are similar. The only difference is that **shobj_description** is used for shared objects. Some system catalogs are global to all databases within each cluster, and the comments for objects in them are stored globally as well.

21.7.26.7 Transaction IDs and Snapshots

The following functions provide server transaction information in an exportable form. The main use of these functions is to determine which transactions were committed between two snapshots.

pgxc_is_committed(transaction_id)

Description: Determines whether the given XID is committed or ignored. NULL indicates the unknown status (such as running, preparing, and freezing).

Return type: Boolean

txid_current()

Description: Gets current transaction ID.

Return type: bigint

txid_current_snapshot()

Description: Gets current snapshot.

Return type: txid_snapshot

txid_snapshot_xip(txid_snapshot)

Description: Gets in-progress transaction IDs in snapshot.

Return type: setof bigint

txid_snapshot_xmax(txid_snapshot)

Description: Gets **xmax** of snapshot.

Return type: bigint

txid_snapshot_xmin(txid_snapshot)

Description: Gets **xmin** of snapshot.

Return type: bigint

txid_visible_in_snapshot(bigint, txid_snapshot)

Description: Queries whether the transaction ID is visible in snapshot. (do not use with subtransaction ids)

Return type: boolean

The internal transaction ID type (**xid**) is 32 bits wide and wraps around every 4 billion transactions. **txid_snapshot**, the data type used by these functions, stores information about transaction ID visibility at a particular moment in time. [Table 21-50](#) describes its components.

Table 21-50 Snapshot components

| Component | Description |
|-----------|--|
| xmin | Earliest transaction ID (txid) that is still active. All earlier transactions will either be committed and visible, or rolled back. |
| xmax | First as-yet-unassigned txid. All txids greater than or equal to this are not yet started as of the time of the snapshot, so they are invisible. |
| xip_list | Active txids at the time of the snapshot. The list includes only those active txids between xmin and xmax ; there might be active txids higher than xmax . A txid that is xmin ≤ txid < xmax and not in this list was already completed at the time of the snapshot, and is either visible or dead according to its commit status. The list does not include txids of subtransactions. |

txid_snapshot's textual representation is **xmin:xmax:xip_list**.

For example: **10:20:10,14,15** means **xmin=10**, **xmax=20**, **xip_list=10, 14, 15**.

21.7.26.8 Computing Node Group Function

pv_compute_pool_workload()

Description: Load status of a computing Node Group.

Return type: void

Example:

```
SELECT * from pv_compute_pool_workload();
nodename | rpinuse | maxrp | nodestate
-----+-----+-----+-----
datanode1 | 0 | 1000 | normal
datanode2 | 0 | 1000 | normal
(2 rows)
```

21.7.26.9 Lock Information Function

pgxc_get_lock_conflicts()

Description: Obtains information about conflicting locks in the cluster. When a lock is waiting for another lock or another lock is waiting for it, a lock conflict occurs.

Return type: SETOF record

Example:

```
SELECT * from pgxc_get_lock_conflicts();
           pgxc_get_lock_conflicts
-----
(relation,cn_5001,gaussdb,u1,test,,,,omm,648406,,, "<insufficient
privilege>",140651661784832,ExclusiveLock,f,2024-10-29 17:24:13.425672+08,)
(1 row)
```

21.7.27 System Management Functions

21.7.27.1 Configuration Settings Functions

Configuration setting functions are used for querying and modifying configuration parameters during running.

current_setting(setting_name)

Description: Specifies the current setting.

Return type: text

Note: **current_setting** obtains the current setting of **setting_name** by query. It is equivalent to the **SHOW** statement. For example:

```
SELECT current_setting('datestyle');
           current_setting
-----
ISO, MDY
(1 row)
```

read_global_var(global_setting_name)

Description: Reads the current value of a global variable.

Return type: text

Note: **read_global_var** is used query the current value of **global_setting_name**. It is equivalent to the **SHOW** statement.

Example:

```
SET my.var = 1000;
SELECT read_global_var('my.var');
           read_global_var
-----
1000
(1 row)
```

 NOTE

global_setting_name indicates a variable similar to **my.var**. The variable name contains decimal points and the left and right sides of the decimal points are not empty. Variables of this type are called global variables.

Global variables can be set only by running the **SET** command. Methods such as **gs_guc**, **ALTER DATABASE dbname SET paraname TO value**, and **ALTER USER username SET paraname TO value** are not supported. Global variables are visible to all users.

set_config(setting_name, new_value, is_local)

Description: Sets the parameter and returns a new value.

Return type: text

Note: **set_config** sets the parameter **setting_name** to **new_value**. If **is_local** is **true**, the new value will only apply to the current transaction. If you want the new value to apply for the current session, use **false** instead. The function corresponds to the **SET** statement. For example:

```
SELECT set_config('log_statement_stats', 'off', false);

set_config
-----
off
(1 row)
```

 CAUTION

If **behavior_compat_options** is set to **DISABLE_SET_GLOBAL_VAR_ON_DATANODE**, you cannot use this function to set global variables on DNs.

21.7.27.2 Universal File Access Functions

Universal file access functions provide local access interfaces for files on a database server. Only files in the database cluster directory and the **log_directory** directory can be accessed. Use a relative path for files in the cluster directory, and a path matching the **log_directory** configuration setting for log files. Only database system administrators can use these functions.

pg_ls_dir(dirname text)

Description: Lists files in a directory.

Return type: setof text

Note: **pg_ls_dir** returns all the names in the specified directory, except the special entries "." and "..".

Examples:

```
SELECT pg_ls_dir('./');

pg_ls_dir
-----
.postgresql.conf.swp
postgresql.conf
```

```
pg_tblspc
PG_VERSION
pg_ident.conf
core
server.crt
pg_serial
pg_twophase
postgresql.conf.lock
pg_stat_tmp
pg_notify
pg_subtrans
pg_ctl.lock
pg_xlog
pg_clog
base
pg_snapshots
postmaster.opts
postmaster.pid
server.key.rand
server.key.cipher
pg_multixact
pg_errorinfo
server.key
pg_hba.conf
pg_replslot
.pg_hba.conf.swp
cacert.pem
pg_hba.conf.lock
global
gaussdb.state
(32 rows)
```

pg_read_file(filename text, offset bigint, length bigint)

Description: Returns the content of a text file.

Return type: text

Note: **pg_read_file** returns part of a text file. It can return a maximum of *length* bytes from *offset*. The actual size of fetched data is less than *length* if the end of the file is reached first. If **offset** is negative, it is the length rolled back from the file end. If **offset** and **length** are omitted, the entire file is returned.

Examples:

```
SELECT pg_read_file('postmaster.pid',0,100);
       pg_read_file
-----
53078          +
/srv/BigData/hadoop/data1/coordinator+
1500022474      +
8000           +
/var/run/dws    +
localhost      +
2
(1 row)
```

pg_read_binary_file(filename text [, offset bigint, length bigint,missing_ok boolean])

Description: Returns the content of a binary file.

Return type: bytea

Note: **pg_read_binary_file** is similar to **pg_read_file**, except that the result is a **bytea** value; accordingly, no encoding checks are performed. In combination with

the **convert_from** function, this function can be used to read a file in a specified encoding:

```
SELECT convert_from(pg_read_binary_file('filename'), 'UTF8');
```

pg_stat_file(filename text)

Description: Returns status information about a file.

Return type: record

Note: **pg_stat_file** returns a record containing the file size, last access timestamp, last modification timestamp, last file status change timestamp, and a **boolean** value indicating if it is a directory. Typical use cases are as follows:

```
SELECT * FROM pg_stat_file('filename');  
SELECT (pg_stat_file('filename')).modification;
```

Examples:

```
SELECT * FROM pg_stat_file('postmaster.pid');  
  
size |      access      |      modification      |      change  
| creation | isdir  
-----+-----+-----+-----  
117 | 2017-06-05 11:06:34+08 | 2017-06-01 17:18:08+08 | 2017-06-01 17:18:08+08  
|      | f  
(1 row)  
SELECT (pg_stat_file('postmaster.pid')).modification;  
      modification  
-----  
2017-06-01 17:18:08+08  
(1 row)
```

21.7.27.3 Server Signaling Functions

Server signaling functions send control signals to other server processes. Only system administrators can use these functions.

pg_cancel_backend(pid int)

Description: Cancels a current query of the backend.

Return type: boolean

Note: **pg_cancel_backend** sends a query cancellation signal (**SIGINT**) to the backend process identified by **pid**. The PID of an active backend process can be found in the **pid** column of the **pg_stat_activity** view, or can be found by listing the database process using **ps** on the server.

Example:

```
SELECT pid FROM pg_stat_activity WHERE stmt_type = 'RESET';  
      pid  
-----  
281471222065200  
(1 row)  
  
SELECT pg_cancel_backend(281471222065200);  
      pg_cancel_backend  
-----  
t  
(1 row)
```

pg_reload_conf()

Description: Notifies the server process to reload the configuration file.

Return type: boolean

Note: **pg_reload_conf** sends a **SIGHUP** signal to the server. As a result, all server processes reload their configuration files.

Example:

```
SELECT pg_reload_conf();
pg_reload_conf
-----
t
(1 row)
```

pg_rotate_logfile()

Description: Rotates the log files of the server.

Return type: boolean

Note: **pg_rotate_logfile** instructs the log file manager to immediately switch to a new output file. This function is valid only if the built-in log collector is running.

Example:

```
SELECT pg_rotate_logfile();
pg_rotate_logfile
-----
t
(1 row)
```

pg_terminate_backend(pid int)

Description: Terminates a backend thread.

Return type: boolean

Note: Each of these functions returns **true** if they are successful and **false** otherwise.

Example:

```
SELECT pid FROM pg_stat_activity;
 pid
-----
140657876268816
140433774061312
140433587902208
140433656592128
140433723717376
140433637189376
140433552770816
140433481983744
140433349310208
(9 rows)

SELECT pg_terminate_backend(140657876268816);
pg_terminate_backend
-----
t
(1 row)
```


pg_wlm_jump_queue(pid int)

Description: Moves a task to the top of the CN queue.

Return type: Boolean

Note: Each of these functions returns **true** if they are successful and **false** otherwise.

Example:

```
SELECT pid FROM pg_stat_activity WHERE stmt_type='RESET';
      pid
-----
281471222065200
(1 row)

SELECT pg_wlm_jump_queue(281471222065200);
 pg_wlm_jump_queue
-----
t
(1 row)
```

gs_wlm_switch_cgroup(pid int, cgroup text)

Description: Moves a job to other Cgroup to improve the job priority.

Return type: Boolean

Note: Each of these functions returns **true** if they are successful and **false** otherwise.

pg_cancel_query(queryId int)

Description: Cancels a current query of the backend. This function is supported in 8.1.2 or later.

Return type: boolean

Note: **pg_cancel_query** sends a query cancellation signal (**SIGINT**) to the backend process identified by **query_id**. The **query_id** of an active backend process can be found in the **query_id** column of the **pg_stat_activity** view.

Example:

```
SELECT query_id FROM pgxc_stat_activity WHERE stmt_type='RESET';
      query_id
-----
0
0
(2 rows)

SELECT pg_cancel_query(0);
 pg_cancel_query
-----
f
(1 row)
```

pgxc_cancel_query(queryId int)

Description: Cancels the query that is being executed in the current cluster. This function is supported in 8.1.2 or later.

Return type: boolean

Note: If the queries of all nodes have been canceled, **true** is returned. Otherwise, **false** is returned.

Example:

```
SELECT query_id FROM pgxc_stat_activity WHERE stmt_type = 'RESET';
query_id
-----
      0
      0
(2 rows)

SELECT pgxc_cancel_query(0);
pgxc_cancel_query
-----
f
(1 row)
```

pg_terminate_query(queryId int)

Description: Terminates a current query of the backend. This function is supported in 8.1.2 or later.

Return type: boolean

Example:

```
SELECT query_id FROM pgxc_stat_activity WHERE stmt_type = 'RESET';
query_id
-----
      0
      0
(2 rows)

SELECT pg_terminate_query(0);
pg_terminate_query
-----
f
(1 row)
```

pgxc_terminate_query(queryId int)

Description: Terminates the query that is being executed in the current cluster. This function is supported in 8.1.2 or later.

Return type: boolean

Example:

```
SELECT query_id FROM pgxc_stat_activity;
query_id
-----
72339069014638631
(1 rows)

SELECT pgxc_terminate_query(72339069014638631);
pgxc_terminate_query
-----
t
(1 row)
```

21.7.27.4 Snapshot Synchronization Functions

Snapshot synchronization functions save the current snapshot and return its identifier.

create_wdr_snapshot()

Description: Creates a performance data snapshot.

Return type: text

NOTE

- Only the database administrator **SYSADMIN** can execute this function.
- This function can be executed only on CNs. If it is executed on DN, the following message will be returned: "WDR snapshot can only be created on coordinator."
- Before executing this function, ensure that the value of **enable_wdr_snapshot** is **on**. If its value is **off**, the following message will be returned for this function: "WDR snapshot request can't be executed, because GUC parameter 'enable_wdr_snapshot' is off."
- If the snapshot thread is not started for some reason, for example, the node is restarted, the following message will be returned for this function: "WDR snapshot request can not be accepted, please retry later."
- If this function fails to be executed, the following message will be returned: "Cannot respond to WDR snapshot request."
- If this function is successfully executed, the following message will be returned: "WDR snapshot request has been submitted." This message indicates that the snapshot creation request has been sent to the background snapshot thread, but does not mean that the snapshot has been successfully created.

kill_snapshot(scope cstring)

Description: Kills the background snapshot thread. This function sends a command to the background snapshot thread and waits for the thread to stop.

The input parameter **scope** indicates the operation scope. Its value can be **local** or **global**.

- Value **local** indicates killing the snapshot thread on the current CN.
- Value **global** indicates killing the snapshot thread on the current CN as well as those on all the other CNs in the cluster.
- If any other value is passed, error message "Scope is invalid, use "local" or "global"." is displayed.
- The input parameter can be left empty, in which case the default value **local** will be used.

Return type: none

NOTE

- Only the database administrator **SYSADMIN** can execute this function.
- This function can be executed only on CNs. If it is executed on DN, the following message will be returned: "kill_snapshot can only be executed on coordinator."
- Executing this function sends a kill signal to the background snapshot thread and waits for it to finish. If the snapshot thread is not killed within 100s, the error message "Kill snapshot thread failed" is displayed.

pg_export_snapshot()

Description: Saves the current snapshot and returns its identifier.

Return type: text

Note: **pg_export_snapshot** saves the current snapshot and returns a text string identifying the snapshot. This string must be passed to clients that want to import the snapshot. A snapshot can be imported when the **set transaction snapshot snapshot_id** command is executed. Doing so is possible only when the transaction is set to the **REPEATABLE READ** isolation level. The output of the function cannot be used as the input of **set transaction snapshot**.

Example:

```
SELECT pg_export_snapshot();
pg_export_snapshot
-----
00000000000A7128-1
(1 row)
```

21.7.27.5 Advisory Lock Functions

Advisory lock functions manage advisory locks. These functions are only for internal use currently.

pg_advisory_lock(key bigint)

Description: Obtains an exclusive session-level advisory lock.

Return type: void

Note: **pg_advisory_lock** locks resources defined by an application. The resources can be identified using a 64-bit or two nonoverlapped 32-bit key values. If another session locks the resources, the function blocks the resources until they can be used. The lock is exclusive. Multiple locking requests are pushed into the stack. Therefore, if the same resource is locked three times, it must be unlocked three times so that it is released to another session.

pg_advisory_lock(key1 int, key2 int)

Description: Obtains an exclusive session-level advisory lock.

Return type: void

pg_advisory_lock_shared(key bigint)

Description: Obtains a shared session-level advisory lock.

Return type: void

pg_advisory_lock_shared(key1 int, key2 int)

Description: Obtains a shared session-level advisory lock.

Return type: void

Note: **pg_advisory_lock_shared** works in the same way as **pg_advisory_lock**, except the lock can be shared with other sessions requesting shared locks. Only would-be exclusive lockers are locked out.

pg_advisory_unlock(key bigint)

Description: Releases an exclusive session-level advisory lock.

Return type: boolean

pg_advisory_unlock(key1 int, key2 int)

Description: Releases an exclusive session-level advisory lock.

Return type: boolean

Note: **pg_advisory_unlock** releases the obtained exclusive advisory lock. If the release is successful, the function returns **true**. If the lock was not held, it will return **false**. In addition, a SQL warning will be reported by the server.

pg_advisory_unlock_shared(key bigint)

Description: Releases a shared session-level advisory lock.

Return type: boolean

pg_advisory_unlock_shared(key1 int, key2 int)

Description: Releases a shared session-level advisory lock.

Return type: boolean

Note: **pg_advisory_unlock_shared** works in the same way as **pg_advisory_unlock**, except it releases a shared session-level advisory lock.

pg_advisory_unlock_all()

Description: Releases all advisory locks owned by the current session.

Return type: void

Note: **pg_advisory_unlock_all** releases all advisory locks owned by the current session. The function is implicitly invoked when the session ends even if the client is abnormally disconnected.

pg_advisory_xact_lock(key bigint)

Description: Obtains an exclusive transaction-level advisory lock.

Return type: void

pg_advisory_xact_lock(key1 int, key2 int)

Description: Obtains an exclusive transaction-level advisory lock.

Return type: void

Note: **pg_advisory_xact_lock** works in the same way as **pg_advisory_lock**, except the lock is automatically released at the end of the current transaction and cannot be released explicitly.

pg_advisory_xact_lock_shared(key bigint)

Description: Obtains a shared transaction-level advisory lock.

Return type: void

pg_advisory_xact_lock_shared(key1 int, key2 int)

Description: Obtains a shared transaction-level advisory lock.

Return type: void

Note: **pg_advisory_xact_lock_shared** works in the same way as **pg_advisory_lock_shared**, except the lock is automatically released at the end of the current transaction and cannot be released explicitly.

pg_try_advisory_lock(key bigint)

Description: Obtains an exclusive session-level advisory lock if available.

Return type: boolean

Note: **pg_try_advisory_lock** is similar to **pg_advisory_lock**, except **pg_try_advisory_lock** does not block the resource until the resource is released. **pg_try_advisory_lock** either immediately obtains the lock and returns **true** or returns **false**, which indicates the lock cannot be performed currently.

pg_try_advisory_lock(key1 int, key2 int)

Description: Obtains an exclusive session-level advisory lock if available.

Return type: boolean

pg_try_advisory_lock_shared(key bigint)

Description: Obtains a shared session-level advisory lock if available.

Return type: boolean

pg_try_advisory_lock_shared(key1 int, key2 int)

Description: Obtains a shared session-level advisory lock if available.

Return type: boolean

Note: **pg_try_advisory_lock_shared** is similar to **pg_try_advisory_lock**, except **pg_try_advisory_lock_shared** attempts to obtain a shared lock instead of an exclusive lock.

pg_try_advisory_xact_lock(key bigint)

Description: Obtains an exclusive transaction-level advisory lock if available.

Return type: boolean

pg_try_advisory_xact_lock(key1 int, key2 int)

Description: Obtains an exclusive transaction-level advisory lock if available.

Return type: boolean

Note: **pg_try_advisory_xact_lock** works in the same way as **pg_try_advisory_lock**, except the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly.

pg_try_advisory_xact_lock_shared(key bigint)

Description: Obtains a shared transaction-level advisory lock if available.

Return type: boolean

pg_try_advisory_xact_lock_shared(key1 int, key2 int)

Description: Obtains a shared transaction-level advisory lock if available.

Return type: boolean

Note: **pg_try_advisory_xact_lock_shared** works in the same way as **pg_try_advisory_lock_shared**, except the lock, if acquired, is automatically released at the end of the current transaction and cannot be released explicitly.

21.7.27.6 Replication Functions

A replication function synchronizes logs and data between instances. It is a statistics or operation method provided by the system to implement HA.

NOTE

Replication functions except statistics queries are internal functions. You are not advised to use them directly.

pg_create_logical_replication_slot('slot_name', 'plugin_name')

Description: Creates a logical replication slot. This is no longer used in 9.1.0.200 and later versions. It is reserved for compatibility with earlier versions.

Parameter:

- **slot_name**
Indicates the name of the streaming replication slot.
Value range: a string, supporting only letters, digits, and the following special characters: `_?-`.
- **plugin_name**
Indicates the name of the plugin.
Value range: a string, supporting only **mppdb_decoding**

Return type: name, text

Note: The first return value is the slot name, and the second is the start LSN position for decoding in the logical replication slot.

pg_create_physical_replication_slot ('slot_name', isDummyStandby)

Description: Creates a physical replication slot.

Parameter:

- **slot_name**
Indicates the name of the streaming replication slot.
Value range: a string, supporting only letters, digits, and the following special characters: _?.-
- **isDummyStandby**
Indicates whether the replication slot is the secondary one.
Value range: a boolean value, **true** or **false**

Return type: name, text

Note: The first return value is the slot name, and the second is the start LSN position for decoding in the physical replication slot.

pg_get_replication_slots()

Description: Displays information about all replication slots on the current DN.

Return type: record

The following information is returned:

Table 21-51 pg_get_replication_slots() fields

| Field | Type | Description |
|---------------|---------|---|
| slot_name | text | Replication slot name |
| plugin | name | Name of the output plug-in of the logical replication slot |
| slot_type | text | Replication slot type |
| datoid | oid | Replication slot's database OID |
| active | boolean | Whether the replication slot is active |
| xmin | xid | Transaction ID of the replication slot |
| catalog_xmin | text | ID of the earliest-decoded transaction corresponding to the logical replication slot. |
| restart_lsn | text | Xlog file information on the replication slot. |
| dummy_standby | boolean | Indicates whether the replication slot is the secondary one. |

Example:

```
SELECT * FROM pg_get_replication_slots();
 slot_name | plugin | slot_type | datoid | active | xmin | catalog_xmin | restart_lsn | dummy_standby
-----+-----+-----+-----+-----+-----+-----+-----+-----
gs_roach_common |      | physical |    0 | f      |      |      | 602861775 | FFFFFFFF/FFFFFFF | f
(1 row)
```

pg_drop_replication_slot('slot_name')

Description: Deletes a streaming replication slot.

Parameter:

slot_name: name of a streaming replication slot. Value range: a string. Only letters, digits, underscores (_), question marks (?), hyphens (-), and periods (.) are allowed.

Return type: void

pg_logical_slot_peek_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

Description: Performs decoding but does not go to the next streaming replication slot. (The decoding result will be returned again on future calls.) This is no longer used in 9.1.0.200 and later versions. It is reserved for compatibility with earlier versions.

Parameter:

- **slot_name**
Indicates the name of the streaming replication slot.
Value range: a string, supporting only letters, digits, and the following special characters: _?-.
- **LSN**
Indicates a target LSN. Decoding is performed only when an LSN is less than or equal to this value.
Value range: a string, in the format of xlogid/xrecoff, for example, '1/2AAFC60' (If this parameter is set to **NULL**, the target LSN indicating the end position of decoding is not specified.)
- **upto_nchanges**
Indicates the number of decoded records (including the **begin** and **commit** timestamps). Assume that there are three transactions, which involve 3, 5, and 7 records, respectively. If **upto_nchanges** is **4**, 8 records of the first two transactions will be decoded. Specifically, decoding is stopped when the number of decoded records exceeds 4 after decoding in the first two transactions is finished.
Value range: a non-negative integer

NOTE

If any of the **LSN** and **upto_nchanges** values are reached, decoding ends.

- options (optional)
 - include-xids
Indicates whether the decoded **data** column contains XID information.
Valid value: **0** and **1**. The default value is **1**.
 - **0**: The decoded **data** column does not contain XID information.
 - **1**: The decoded **data** column contains XID information.
 - skip-empty-xacts
Indicates whether to ignore empty transaction information during decoding.
Valid value: **0** and **1**. The default value is **0**.
 - **0**: The empty transaction information is not ignored during decoding.
 - **1**: The empty transaction information is ignored during decoding.
 - include-timestamp
Indicates whether decoding information contains the **commit** timestamp.
Valid value: **0** and **1**. The default value is **0**.
 - **0**: The decoding information does not contain the **commit** timestamp.
 - **1**: The decoding information contains the **commit** timestamp.

Return type: text, uint, text

Note: The function returns the decoding result. Each decoding result contains three columns, corresponding to the above return types and indicating the LSN position, XID, and decoded content, respectively.

pg_logical_slot_get_changes('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value')

Description: Performs decoding and goes to the next streaming replication slot. This is no longer used in 9.1.0.200 and later versions. It is reserved for compatibility with earlier versions.

Parameter: This function has the same parameters as **pg_logical_slot_peek_changes**. For details, see [pg_logical_slot_peek_changes\('slot_name', 'LSN', upto_nchanges, 'options_name', 'options_value'\)](#).

pg_replication_slot_advance ('slot_name', 'LSN')

Description: Directly goes to the streaming replication slot for a specified LSN, without outputting any decoding result.

Parameter:

- slot_name
Indicates the name of the streaming replication slot.

Value range: a string, supporting only letters, digits, and the following special characters: _?-..

- LSN

Indicates a target LSN. Next decoding will be performed only in transactions whose commission position is greater than this value. If an input LSN is smaller than the position recorded in the current streaming replication slot, the function directly returns. If the input LSN is greater than the LSN of the current physical log, the latter LSN will be directly used for decoding.

Value range: a string, in the format of xlogid/xrecoff

Return type: name, text

Note: A return result contains the slot name and LSN that is actually used for decoding.

pg_stat_get_data_senders()

Description: Displays statistics about replication sending threads on all data page on the current DN.

Return type: record

The following information is returned:

Table 21-52 pg_stat_get_data_senders() fields

| Field | Type | Description |
|------------------|--------------------------|--|
| pid | bigint | Thread PID |
| sender_pid | integer | Current sender PID |
| local_role | text | Local role |
| peer_role | text | Peer role |
| state | text | Current sender's replication status |
| catchup_start | timestamp with time zone | Startup time of a catchup task |
| catchup_end | timestamp with time zone | End time of a catchup task |
| queue_size | text | Data queue size |
| queue_lower_tail | text | Position of data queue tail 1 |
| queue_header | text | Position of data queue header |
| queue_upper_tail | text | Position of data queue tail 2 |
| send_position | text | Sending position of the sender |
| receive_position | text | Receiving position of the receiver |
| catchup_type | text | Catchup task type, full or incremental |

| Field | Type | Description |
|------------------------|---------|--|
| catchup_bcm_filename | text | BCM file executed by the current catchup task |
| catchup_bcm_finished | integer | Number of BCM files completed by a catchup task |
| catchup_bcm_total | integer | Total number of BCM files to be operated by a catchup task |
| catchup_percent | text | Completion percentage of a catchup task |
| catchup_remaining_time | text | Estimated remaining time of a catchup task |

pg_stat_get_wal_senders()

Description: Displays statistics about replication sending threads on all WALs on the current DN.

Return type: record

The following information is returned:

Table 21-53 pg_stat_get_wal_senders() fields

| Field | Type | Description |
|-----------------------|--------------------------|--|
| pid | bigint | Thread PID |
| sender_pid | integer | Current sender PID |
| local_role | text | Local role |
| peer_role | text | Peer role |
| peer_state | text | Peer status |
| state | text | Current sender's replication status |
| catchup_start | timestamp with time zone | Startup time of a catchup task |
| catchup_end | timestamp with time zone | End time of a catchup task |
| sender_sent_location | text | Location where the sender sends LSNs |
| sender_write_location | text | Location where the sender writes LSNs |
| sender_flush_location | text | Location where the sender flushes LSNs |

| Field | Type | Description |
|----------------------------|---------|---|
| sender_replay_location | text | Location where the sender replays LSNs |
| receiver_received_location | text | Location where the receiver receives LSNs |
| receiver_write_location | text | Location where the receiver writes LSNs |
| receiver_flush_location | text | Location where the receiver flushes LSNs |
| receiver_replay_location | text | Location where the receiver replays LSNs |
| sync_percent | text | Specifies the synchronization percentage. |
| sync_state | text | Synchronization state (asynchronous duplication, synchronous duplication, or potential synchronization) |
| sync_priority | integer | Priority of synchronous duplication (0 indicates asynchronization) |
| sync_most_available | text | Whether to block the active node when the synchronization on the standby node fails |
| channel | text | WALSender channel information |

pg_stat_get_wal_receiver()

Description: Displays statistics about replication receiving threads on all WALs on the current DN.

Return type: record

The following information is returned:

Table 21-54 pg_stat_get_wal_receiver()

| Field | Type | Description |
|--------------|---------|---------------------------------------|
| receiver_pid | integer | Current receiver PID |
| local_role | text | Local role |
| peer_role | text | Peer role |
| peer_state | text | Peer status |
| state | text | Current receiver's replication status |

| Field | Type | Description |
|----------------------------|------|---|
| sender_sent_location | text | Location where the sender sends LSNs |
| sender_write_location | text | Location where the sender writes LSNs |
| sender_flush_location | text | Location where the sender flushes LSNs |
| sender_replay_location | text | Location where the sender replays LSNs |
| receiver_received_location | text | Location where the receiver receives LSNs |
| receiver_write_location | text | Location where the receiver writes LSNs |
| receiver_flush_location | text | Location where the receiver flushes LSNs |
| receiver_replay_location | text | Location where the receiver replays LSNs |
| sync_percent | text | Specifies the synchronization percentage. |
| channel | text | WALReceiver channel information |

pg_stat_get_stream_replications()

Description: Displays information about all replication statistics on the current DN.

Return type: record

The following information is returned:

Table 21-55 pg_stat_get_stream_replications()

| Field | Type | Description |
|--------------------|---------|-----------------------|
| local_role | text | Local role |
| static_connections | integer | Connection statistics |
| db_state | text | Database status |
| detail_information | text | Detail information |

Example:

```
SELECT * FROM pg_stat_get_stream_replications();  
local_role | static_connections | db_state | detail_information
```

```
-----+-----+-----+-----  
Normal |          0 | Normal | Normal  
(1 row)
```

pg_stat_xlog_space()

Description: Displays the Xlog space usage on the current DN.

Return type: record

The following information is returned:

Table 21-56 pg_stat_xlog_space()

| Column | Type | Description |
|------------|--------|---|
| xlog_files | bigint | Number of all identified xlog files in the pg_xlog directory, excluding the backup and archive_status subdirectories. |
| xlog_size | bigint | Total size (MB) of all identified xlog files in the pg_xlog directory, excluding the backup and archive_status subdirectories. |
| other_size | bigint | Total size (MB) of files in the backup and archive_status subdirectories of the pg_xlog directory. |

Example:

```
SELECT * FROM pg_stat_xlog_space();  
xlog_files | xlog_size | other_size  
-----+-----+-----  
       79 |      1264 |         0  
(1 row)
```

pgxc_stat_xlog_space()

Description: Displays the Xlog space usage on all active DNs.

Return type: record

The following information is returned:

Table 21-57 pgxc_stat_xlog_space()

| Column | Type | Description |
|-----------|------|-------------|
| node_name | name | Node name |

| Column | Type | Description |
|------------|--------|---|
| xlog_files | bigint | Number of all identified xlog files in the pg_xlog directory, excluding the backup and archive_status subdirectories. |
| xlog_size | bigint | Total size (MB) of all identified xlog files in the pg_xlog directory, excluding the backup and archive_status subdirectories. |
| other_size | bigint | Total size (MB) of files in the backup and archive_status subdirectories of the pg_xlog directory. |

Example:

```
SELECT * FROM pgxc_stat_xlog_space();
 node_name | xlog_files | xlog_size | other_size
-----+-----+-----+-----
dn_6001_6002 |      73 |    1168 |         0
dn_6003_6004 |      73 |    1168 |         0
dn_6005_6006 |      73 |    1168 |         0
cn_5003      |      79 |    1264 |         0
cn_5001      |      72 |    1152 |         0
cn_5002      |      73 |    1168 |         0
(6 rows)
```

21.7.27.7 Resource Management Functions

This section describes the functions of the resource management module.

gs_switch_respool(query_id bigint, resource_pool_name name)

Description: Job resource pool switchover function is supported by cluster versions 9.1.0 and later.

The value of **resource_pool_name** must contain less than 64 characters. Upon calling this function, the system will decide whether to start a job in the queuing phase when switching to the new resource pool. If a job is running, the system will switch to the new resource pool to start the queuing job from the original resource pool. Additionally, the CPU control of the new resource pool will limit the queuing job.

This function supports resource pool switchover in both static and dynamic management and control modes, as well as resource pool switchover using short queries.

Return type: bool

CAUTION

- Only administrators or users with administrator permissions can use this function. Common users are not authorized to use it. To use this function, the cluster's CPU control function, specifically the **Cgroup** function, must be functioning normally. You can use the **pgxc_cgroup_reload_conf()** function to check whether **Cgroup** is normal. If the command output contains non-success information, rectify the fault.
- After the resource pool is switched, the function performs management and control based on the exception rules of the new resource pool.
- If a job is in the global concurrent queue, which means it is waiting in the global queue, it will not take effect right away after the resource pool is switched. At this point, the job has not entered the resource pool queue yet.
- 1. Job CPU control switchover supports CNs and DNs. For example, if resource pool A uses 40 cores and resource pool B uses 5 cores, switching a job from pool A to pool B will execute the job on the 5 cores of pool B. This results in slower execution but isolates the job, ensuring that jobs in pool A are unaffected, thus degrading the statement.
- When switching resource pools, use similar statements to perform batch switchovers. Ensure that the jobs being switched are in the same state, such as all queuing or all running. If switching between different types of statements, it is recommended to switch them twice. Switching them together may exceed the concurrent statement limit, causing abnormal job wake-ups.

```
SELECT gs_switch_respool (queryid,'xxx') FROM xxx where xxx;
```
- The restrictions on switching the statement resource pool are as follows:
 - If jobs from resource pool A are switched to pool B, jobs at the head of the queue in pool A will be woken up. If the concurrent job limit in pool B is reached, job A1 will not run immediately.
 - Switch job A2 from queuing in pool A to pool B. If the concurrent job limit in pool B is reached, job A2 will continue to queue. If not, job A2 will wake up and run immediately.
 - During resource pool switchover, only the same lane can be switched. For example, switching the fast lane of pool A will switch the fast lane of pool B. If the short query acceleration function is not enabled in pool B, the job runs directly regardless of its status in pool B.
- This function can be used only when all nodes are normal. If a node is faulty, the function fails to be executed.

Example:

While executing the statement with ID **74590868828368101**, the administrator successfully switches it to the **poolg1** resource pool.

```
CALL gs_switch_respool(74590868828368101,'poolg1');
gs_switch_respool
-----
t
(1 row)
```

gs_increase_except_num(unique_sql_id int8)

Description: Records job exception information. The input parameter must be greater than 0. If this function is invoked, the number of job exceptions will be

increased by 1 and the latest job exception time will be updated to the current time.

Return type: bool

Example:

```
SELECT gs_increase_except_num(111);
gs_increase_except_num
-----
t
(1 row)
```

gs_increase_except_num(unique_sql_id int8, except_num int4)

Description: Records job exception information. The input parameter must be greater than 0. If this function is invoked, the number of job exceptions will be increased by **except_num** and the latest job exception time will be updated to the current time.

Return type: bool

Example:

```
SELECT gs_increase_except_num(111, 4);
gs_increase_except_num
-----
t
(1 row)
```

gs_increase_except_num(unique_sql_id int8, except_num int4, except_time int8)

Description: Records job exception information. The input parameter must be greater than 0. If this function is invoked, the number of job exceptions will be increased by **except_num** and the latest job exception time will be updated to **except_time**, a timestamp. This function is mainly invoked internally.

Return type: bool

Example:

```
SELECT gs_increase_except_num(111, 4, 714623414421256);
gs_increase_except_num
-----
t
(1 row)
```

gs_append_blocklist(unique_sql_id int8)

Description: Adds a job to the blocklist and update the blocklist information in **GS_BLOCKLIST_QUERY**.

Return type: bool

Example:

```
SELECT gs_append_blocklist(111);
gs_append_blocklist
-----
t
(1 row)
```

gs_remove_blocklist(unique_sql_id int8)

Description: Removes a job from the blocklist. and update the blocklist information in **GS_BLOCKLIST_QUERY**.

Return type: bool

Example:

```
SELECT gs_remove_blocklist(111);
gs_append_blocklist
-----
t
(1 row)
```

gs_increase_sql_except_num(sql_hash text)

Description: Records job exception information. If this function is invoked, the number of job exceptions will be increased by 1 and the latest job exception time will be updated to the current time. This function is supported only by 9.1.0.200 and later cluster versions and is used for internal calling.

Return type: bool

Example:

```
SELECT gs_increase_sql_except_num('sql_7bf6381d2a5a456a1936027198ad8b12');
gs_increase_sql_except_num
-----
t
(1 row)
```

gs_increase_sql_except_num(sql_hash text, except_num int4)

Description: Records job exception information. If this function is invoked, the number of job exceptions will be increased by **except_num** and the latest job exception time will be updated to the current time. This function is supported only by 9.1.0.200 and later cluster versions and is used for internal calling.

Return type: bool

Example:

```
SELECT gs_increase_sql_except_num('sql_7bf6381d2a5a456a1936027198ad8b12', 4);
gs_increase_sql_except_num
-----
t
(1 row)
```

gs_increase_sql_except_num(sql_hash text, except_num int4, except_time int8)

Description: Records job exception information. If this function is invoked, the number of job exceptions will be increased by **except_num** and the latest job exception time will be updated to **except_time**. **except_time** functions as a timestamp here. This function is supported only by 9.1.0.200 and later cluster versions and is used for internal calling.

Return type: bool

Example:

```
SELECT gs_increase_sql_except_num('sql_7bf6381d2a5a456a1936027198ad8b12', 4, 714623414421256);
gs_increase_sql_except_num
-----
t
(1 row)
```

gs_append_blocklist(sql_hash text)

Description: Adds a job to the blocklist and updates the blocklist information in **GS_BLOCKLIST_SQL**. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: bool

Example:

```
SELECT gs_append_blocklist('sql_7bf6381d2a5a456a1936027198ad8b12');
gs_append_blocklist
-----
t
(1 row)
```

gs_remove_blocklist(sql_hash text)

Description: Removes a job from the blocklist and updates the blocklist information in **GS_BLOCKLIST_SQL**. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: bool

Example:

```
SELECT gs_remove_blocklist('sql_7bf6381d2a5a456a1936027198ad8b12');
gs_remove_blocklist
-----
t
(1 row)
```

gs_wlm_rebuild_except_rule_hash()

Description: Rebuilds the memory hash table of exception rules on the current node. The monitoring thread obtains the exception rule thresholds from the memory hash table in real time. When the hash table is abnormal, this function can be used to rebuild the table.

Return type: bool

Example:

```
SELECT gs_wlm_rebuild_except_rule_hash();
gs_wlm_rebuild_except_rule_hash
-----
t
(1 row)
```

gs_wlm_readjust_user_space(oid)

Description: This function calibrates the permanent storage space of a user. The input parameter is the user OID. If the input parameter is set to **0**, the permanent storage space of all users is calibrated.

Return type: text

Example:

```
SELECT gs_wlm_readjust_user_space(0);  
gs_wlm_readjust_user_space  
-----  
Exec Success  
(1 row)
```

pgxc_wlm_readjust_schema_space()

Description: This function calibrates the permanent storage space of a schema.

Return type: text

Example:

```
SELECT pgxc_wlm_readjust_schema_space();  
pgxc_wlm_readjust_schema_space  
-----  
Exec Success  
(1 row)
```

pgxc_wlm_readjust_relfilenode_size_table()

Description: Space statistics calibration function. It does not recreate the **PG_RELFILENODE_SIZE** system catalog but recalibrates the user and schema space.

NOTE

Due to transaction isolation between the calibration function and other services, the calibration function is invisible to other services that are being executed. As a result, the calibration function does not involve space changes of such services. To avoid space difference errors after calibration, you are advised to use the space calibration function to perform calibration when the space is stable.

Return type: text

Example:

```
SELECT pgxc_wlm_readjust_schema_space();  
pgxc_wlm_readjust_relfilenode_size_table  
-----  
Exec Success  
(1 row)
```

pgxc_wlm_readjust_relfilenode_size_table(integer)

Description: Space statistics calibration function

NOTE

Due to transaction isolation between the calibration function and other services, the calibration function is invisible to other services that are being executed. As a result, the calibration function does not involve space changes of such services. To avoid space difference errors after calibration, you are advised to use the space calibration function to perform calibration when the space is stable.

Parameter: The value ranges from 0 to 4. Different input parameter values indicate different calibration granularities.

- If the input parameter is set to **0** (default value), the **PG_RELFILENODE_SIZE** system catalog is not rebuilt and the user and schema space are recalibrated.

- If the input parameter is set to **1**, the **PG_RELFILENODE_SIZE** system catalog is rebuilt and the user and schema space are recalibrated.
- If the input parameter is set to **2**, the **PG_RELFILENODE_SIZE** system catalog is rebuilt.
- If the input parameter is set to **3**, the schema space is recalibrated.
- If the input parameter is set to **4**, the user space is recalibrated.

Return type: text

Example:

```
SELECT * FROM pgxc_wlm_readjust_relfilenode_size_table(1);
      result
-----
Exec success
(1 row)
```

pgxc_wlm_get_schema_space(cstring)

Description: Obtains the schema space of each instance in a specified logical cluster on the CN.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|--------------|--------|--------------------------|
| schemaname | text | Schema name |
| schemaid | oid | Schema OID |
| databasename | text | Database name |
| databaseid | oid | Database OID |
| nodename | text | Instance name |
| nodegroup | text | Name of the node group |
| usedspace | bigint | Size of the used space |
| permSPACE | bigint | Upper limit of the space |

Example:

```
SELECT * FROM pgxc_wlm_get_schema_space('group1');
 schemaname | schemaid | databasename | databaseid | nodename | nodegroup | usedspace | permSPACE
-----+-----+-----+-----+-----+-----+-----+-----
pg_catalog  |      11 | test1       | 16384 | datanode1 | installation | 9469952 | -1
public      |     220 | postgres    | 15253 | datanode1 | installation | 25280512 | -1
pg_toast    |      99 | test1       | 16384 | datanode1 | installation | 1859584 | -1
cstore      |     100 | test1       | 16384 | datanode1 | installation | 0 | -1
data_redis  |    18106 | postgres    | 15253 | datanode1 | installation | 655360 | -1
data_redis  |    18116 | test1       | 16384 | datanode1 | installation | 0 | -1
public      |     220 | test1       | 16384 | datanode1 | installation | 16384 | -1
dbms_om     |     3987 | postgres    | 15253 | datanode1 | installation | 0 | -1
dbms_job    |     3988 | postgres    | 15253 | datanode1 | installation | 0 | -1
```

| | | | | | | | | | | | | | | |
|--------------------|--|-------|--|----------|--|-------|--|-----------|--|--------------|--|----------|--|----|
| dbms_om | | 3987 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| dbms_job | | 3988 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| sys | | 11693 | | postgres | | 15253 | | datanode1 | | installation | | 0 | | -1 |
| sys | | 11693 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| utl_file | | 14644 | | postgres | | 15253 | | datanode1 | | installation | | 0 | | -1 |
| utl_raw | | 14669 | | postgres | | 15253 | | datanode1 | | installation | | 0 | | -1 |
| dbms_sql | | 14674 | | postgres | | 15253 | | datanode1 | | installation | | 0 | | -1 |
| dbms_output | | 14662 | | postgres | | 15253 | | datanode1 | | installation | | 0 | | -1 |
| dbms_random | | 14666 | | postgres | | 15253 | | datanode1 | | installation | | 0 | | -1 |
| dbms_lob | | 14701 | | postgres | | 15253 | | datanode1 | | installation | | 0 | | -1 |
| information_schema | | 14300 | | postgres | | 15253 | | datanode1 | | installation | | 294912 | | -1 |
| information_schema | | 14300 | | test1 | | 16384 | | datanode1 | | installation | | 294912 | | -1 |
| utl_file | | 14644 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| dbms_output | | 14662 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| dbms_random | | 14666 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| utl_raw | | 14669 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| dbms_sql | | 14674 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| dbms_lob | | 14701 | | test1 | | 16384 | | datanode1 | | installation | | 0 | | -1 |
| pg_catalog | | 11 | | postgres | | 15253 | | datanode1 | | installation | | 13049856 | | -1 |
| redisuser | | 16387 | | postgres | | 15253 | | datanode1 | | installation | | 630784 | | -1 |
| pg_toast | | 99 | | postgres | | 15253 | | datanode1 | | installation | | 3080192 | | -1 |
| cstore | | 100 | | postgres | | 15253 | | datanode1 | | installation | | 2408448 | | -1 |
| pg_catalog | | 11 | | test1 | | 16384 | | datanode2 | | installation | | 9469952 | | -1 |
| public | | 2200 | | postgres | | 15253 | | datanode2 | | installation | | 25214976 | | -1 |
| pg_toast | | 99 | | test1 | | 16384 | | datanode2 | | installation | | 1859584 | | -1 |
| cstore | | 100 | | test1 | | 16384 | | datanode2 | | installation | | 0 | | -1 |
| data_redis | | 18106 | | postgres | | 15253 | | datanode2 | | installation | | 655360 | | -1 |
| data_redis | | 18116 | | test1 | | 16384 | | datanode2 | | installation | | 0 | | -1 |
| public | | 2200 | | test1 | | 16384 | | datanode2 | | installation | | 16384 | | -1 |
| dbms_om | | 3987 | | postgres | | 15253 | | datanode2 | | installation | | 0 | | -1 |
| dbms_job | | 3988 | | postgres | | 15253 | | datanode2 | | installation | | 0 | | -1 |
| dbms_om | | 3987 | | test1 | | 16384 | | datanode2 | | installation | | 0 | | -1 |
| dbms_job | | 3988 | | test1 | | 16384 | | datanode2 | | installation | | 0 | | -1 |

pgxc_wlm_analyze_schema_space(cstring)

Description: Obtains the schema space of a specified logical cluster on the CN.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|--------------|---------|--|
| schemaname | text | Schema name |
| databasename | text | Database name |
| nodegroup | text | Name of the node group |
| total_value | bigint | Total cluster space in the current schema |
| avg_value | bigint | Average space of instances in the current schema |
| skew_percent | integer | Skew ratio |
| extend_info | text | Extended information, including the maximum space of a single instance, minimum space of a single instance, and names of the instances with the maximum space or minimum space |

Example:

```
SELECT * FROM pgxc_wlm_analyze_schema_space('group1');
  schemaname | databasename | nodegroup | total_value | avg_value | skew_percent |
extend_info
-----+-----+-----+-----+-----+-----+
pg_catalog   | test1        | installation | 56819712 | 9469952 | 0 | min:9469952
datanode1,max:9469952 datanode1
public       | postgres     | installation | 150495232 | 25082538 | 0 | min:24903680
datanode6,max:25280512 datanode1
pg_toast     | test1        | installation | 11157504 | 1859584 | 0 | min:1859584
datanode1,max:1859584 datanode1
cstore      | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
data_redis  | postgres     | installation | 1966080 | 327680 | 50 | min:0
datanode4,max:655360 datanode1
data_redis  | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
public      | test1        | installation | 98304 | 16384 | 0 | min:16384
datanode1,max:16384 datanode1
dbms_om     | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0
datanode1
dbms_job    | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0
datanode1
dbms_om     | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
dbms_job    | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
sys         | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
sys         | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
utl_file    | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
utl_raw     | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
dbms_sql    | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
dbms_output | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0
datanode1
dbms_random | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0
datanode1
dbms_lob    | postgres     | installation | 0 | 0 | 0 | min:0 datanode1,max:0
datanode1
information_schema | postgres | installation | 1769472 | 294912 | 0 | min:294912
datanode1,max:294912 datanode1
information_schema | test1    | installation | 1769472 | 294912 | 0 | min:294912
datanode1,max:294912 datanode1
utl_file    | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
dbms_output | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0
datanode1
dbms_random | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0
datanode1
utl_raw     | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
dbms_sql    | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
dbms_lob    | test1        | installation | 0 | 0 | 0 | min:0 datanode1,max:0 datanode1
pg_catalog  | postgres     | installation | 75431936 | 12571989 | 3 | min:12124160
datanode4,max:13049856 datanode1
redisuser   | postgres     | installation | 1884160 | 314026 | 50 | min:16384
datanode4,max:630784 datanode1
pg_toast    | postgres     | installation | 17154048 | 2859008 | 7 | min:2637824
datanode4,max:3080192 datanode1
cstore     | postgres     | installation | 15294464 | 2549077 | 5 | min:2408448
datanode1,max:2703360 datanode6
(31 rows)
```

gs_wlm_set_queryband_action(cstring,cstring,int4)

Description: Sets the action and query order of **query_band**.

Return type: boolean

The following table describes the input parameters.

| Name | Type | Description |
|--------|---------|---|
| qband | cstring | Query band key-value pair. The maximum length is 63 characters. |
| action | cstring | Action associated to a query band |
| order | int4 | Query band query order. The default value is -1 . |

Example:

```
SELECT * FROM gs_wlm_set_queryband_action('a=1','respool=p1');
gs_wlm_set_queryband_action
-----
t
(1 row)

SELECT * FROM gs_wlm_set_queryband_action('a=3','respool=p1;priority=rush',1);
gs_wlm_set_queryband_action
-----
t
(1 row)
```

gs_wlm_set_queryband_order(cstring,int4)

Description: Sets the **query_band** query order.

Return type: boolean

The following table describes the input parameters.

| Name | Type | Description |
|-------|---------|--|
| qband | cstring | query_band key-value pairs |
| order | int4 | Query band query order. The default value is -1 . |

Example:

```
SELECT * FROM gs_wlm_set_queryband_order('a=1',2);
gs_wlm_set_queryband_order
-----
t
(1 row)
```

gs_wlm_get_queryband_action(cstring)

Description: Obtains the action and query order of **query_band**.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|------------|---------|---|
| qband | cstring | query_band key-value pairs |
| respool_id | Oid | OID of the resource pool associated with query_band |
| respool | text | Name of the resource pool associated with query_band |
| priority | text | Intra-queue priority associated with query_band |
| qborder | int4 | query_band query order |

Example:

```
SELECT * FROM gs_wlm_get_queryband_action('a=1');
qband | respool_id | respool | priority | qborder
-----+-----+-----+-----+-----
a=1   | 16388 | p1     | Medium   | -1
(1 row)
```

gs_cgroup_reload_conf()

Description: This function loads the Cgroup configuration file online on the current instance.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|-----------|------|--|
| node_name | text | Instance name |
| node_host | text | IP address of the node where the instance is located |
| result | text | Whether Cgroup online loading is successful |

Example:

```
SELECT * FROM gs_cgroup_reload_conf();
node_name | node_host | result
-----+-----+-----
cn_5001   | 192.168.178.35 | success
```

pgxc_cgroup_reload_conf()

Description: This function loads the Cgroup configuration file online on all instances of the system.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|-----------|------|--|
| node_name | text | Instance name |
| node_host | text | IP address of the node where the instance is located |
| result | text | Whether Cgroup online loading is successful |

Example:

```
SELECT * FROM pgxc_cgroup_reload_conf();
 node_name | node_host | result
-----+-----+-----
dn_6025_6026 | 192.168.178.177 | success
dn_6049_6050 | 192.168.179.79 | success
dn_6051_6052 | 192.168.179.79 | success
dn_6055_6056 | 192.168.179.79 | success
dn_6067_6068 | 192.168.181.57 | success
dn_6023_6024 | 192.168.178.39 | success
dn_6009_6010 | 192.168.181.21 | success
dn_6011_6012 | 192.168.181.21 | success
dn_6015_6016 | 192.168.181.21 | success
dn_6029_6030 | 192.168.178.177 | success
dn_6031_6032 | 192.168.178.177 | success
dn_6045_6046 | 192.168.179.45 | success
cn_5001      | 192.168.178.35 | success
cn_5003      | 192.168.178.39 | success
dn_6061_6062 | 192.168.181.179 | success
cn_5006      | 192.168.179.45 | success
cn_5004      | 192.168.178.177 | success
cn_5002      | 192.168.181.21 | success
cn_5005      | 192.168.178.187 | success
dn_6019_6020 | 192.168.178.39 | success
dn_6007_6008 | 192.168.178.35 | success
dn_6071_6072 | 192.168.181.57 | success
dn_6003_6004 | 192.168.178.35 | success
dn_6013_6014 | 192.168.181.21 | success
dn_6035_6036 | 192.168.178.187 | success
dn_6037_6038 | 192.168.178.187 | success
dn_6001_6002 | 192.168.178.35 | success
dn_6063_6064 | 192.168.181.179 | success
dn_6005_6006 | 192.168.178.35 | success
dn_6057_6058 | 192.168.181.179 | success
dn_6069_6070 | 192.168.181.57 | success
dn_6027_6028 | 192.168.178.177 | success
dn_6059_6060 | 192.168.181.179 | success
dn_6041_6042 | 192.168.179.45 | success
dn_6043_6044 | 192.168.179.45 | success
dn_6047_6048 | 192.168.179.45 | success
dn_6033_6034 | 192.168.178.187 | success
dn_6065_6066 | 192.168.181.57 | success
dn_6021_6022 | 192.168.178.39 | success
dn_6017_6018 | 192.168.178.39 | success
dn_6039_6040 | 192.168.178.187 | success
dn_6053_6054 | 192.168.179.79 | success
(42 rows)
```

pgxc_cgroup_reload_conf(text)

Description: This function loads the Cgroup configuration file online on a node. The input parameter is the IP address of the node.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|-----------|------|--|
| node_name | text | Instance name |
| node_host | text | IP address of the node where the instance is located |
| result | text | Whether Cgroup online loading is successful |

Example:

```
SELECT * FROM pgxc_cgroup_reload_conf('192.168.178.35');
 node_name | node_host | result
-----+-----+-----
 cn_5001   | 192.168.178.35 | success
 dn_6007_6008 | 192.168.178.35 | success
 dn_6003_6004 | 192.168.178.35 | success
 dn_6001_6002 | 192.168.178.35 | success
 dn_6005_6006 | 192.168.178.35 | success
(5 rows)
```

gs_wlm_node_recover(boolean isForce)

Description: This function updates and restores job information and counts on the CCN in dynamic resource management mode. It can be executed only by administrators, and is usually used to restore a faulty CN after it was restarted. This function is called by the Cluster Manager (CM). Its usage is as follows:

- If this function is executed by CN, it instructs the CCN to clear job information and counts on the CN.
- If this function is executed by CCN, it resets job counts and obtains the latest slow lane job information from the CN.

Return type: bool

gs_wlm_node_clean(cstring nodename)

Description: On the CCN in dynamic resource management mode, clears the job information and counts of a specified CN. This function can be executed only by administrators, and is usually used to restore a faulty CN after it was restarted. This function is called by the Cluster Manager (CM). Generally, users are not advised to call it.

Return type: bool

pg_stat_get_wlm_node_resource_info(int4)

Description: This function displays the summary of all DN resources.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|-----------------|---------|-------------------------------|
| min_mem_util | integer | Minimum memory usage of a DN |
| max_mem_util | integer | Maximum memory usage of a DN |
| min_cpu_util | integer | Minimum CPU usage of a DN |
| max_cpu_util | integer | Maximum CPU usage of a DN |
| min_io_util | integer | Minimum I/O usage of a DN |
| max_io_util | integer | Maximum I/O usage of a DN |
| phy_usemem_rate | integer | Maximum physical memory usage |

pg_stat_get_workload_struct_info()

Description: Load management function for locating CCN queuing problems. This function is an internal function. To use this function, contact technical support.

Return type: record

pgxc_query_resource_info(query_id bigint)

Description: This function displays resource monitoring information about the statement with a specified query ID on all DNs. It is supported only by clusters of version 9.1.0.100 or later.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|------------|--------|--|
| node_name | text | Instance name, which contains only DNs. |
| user_id | oid | User ID. |
| queryid | bigint | Internal query ID used for statement execution. |
| used_mem | int | Memory used by the statement on the current DN. The unit is MB. |
| cpu_time | bigint | CPU time of a statement on the current DN. The unit is ms. |
| used_cpu | double | Number of CPUs used by the statement on the current DN. |
| spill_size | bigint | Amount of data spilled to disks on the current DN. The default value is 0. The unit is MB. |
| read_bytes | bigint | Number of logical read bytes used by the statement on the current DN. The unit is KB. |

| Column | Type | Description |
|-------------|--------|--|
| write_bytes | bigint | Number of logical write bytes used by the statement on the current DN. The unit is KB. |
| read_count | bigint | Number of logical reads used by the statement on the current DN. |
| write_count | bigint | Number of logical writes used by the statement on the current DN. |
| read_speed | int | Logical read rate used by the statement on the current DN. The unit is KB/s. |
| write_speed | int | Logical write rate used by the statement on the current DN. The unit is KB/s. |
| curr_iops | int | I/O operations per second of the statement on the current DN. It is recorded as a count in a column-store table and as a count of 10,000 in a row-store table. |
| send_pkg | bigint | Total number of communication packages sent by a statement across all DNs. |
| recv_pkg | bigint | Total number of communication packages received by a statement across all DNs. |
| send_bytes | bigint | Total sent data of the statement stream, in byte. |
| recv_bytes | bigint | Total received data of the statement stream, in byte. |
| send_speed | int | Network sending rate of the statement on the current DN. The unit is KB/s. |
| recv_speed | int | Network receiving rate of the statement on the current DN. The unit is KB/s. |

21.7.27.8 Memory Management Functions

This function is supported only by cluster versions 9.1.0 and later.

pg_shared_chunk_detail(contextname char(64))

Description: This function queries information about all chunks requested by the memory context in a specified shared memory.

The **contextname** parameter indicates the name of the memory context.

Before using this function, use the **pg_shared_chunk_dump(contextname char(64))** function to print related information to a file.

Return type: record

Example:

```
SELECT * FROM pg_shared_chunk_detail('pgstat file hash table');
   parent | level | file_name | line_number | chunk_size | requested_number | total_size
-----+-----+-----+-----+-----+-----+-----
pgstat file context | 2 | dynahash.cpp | 158 | 2048 | 1 | 2048
pgstat file context | 2 | dynahash.cpp | 158 | 256 | 1 | 256
pgstat file context | 2 | dynahash.cpp | 158 | 4096 | 9 | 36864
pgstat file context | 2 | dynahash.cpp | 158 | 8192 | 4 | 32768
(4 rows)
```

pv_session_chunk_detail(tid int, contextname char(64))

Description: This function queries information about all chunks applied for by a memory context created by a specified thread.

The **tid** parameter indicates the thread ID, while the **contextname** parameter indicates the memory context name.

To use this function, you need to use the **pv_session_chunk_dump(tid int, contextname char(64))** function to print related information to a file.

Return type: record

Example:

```
SELECT * FROM pv_session_chunk_detail(140178810990936, 'Timezones');
   parent | level | file_name | line_number | chunk_size | requested_number | total_size
-----+-----+-----+-----+-----+-----+-----
TopMemoryContext | 1 | dynahash.cpp | 158 | 1280 | 2 | 2560
TopMemoryContext | 1 | dynahash.cpp | 158 | 160 | 1 | 160
TopMemoryContext | 1 | dynahash.cpp | 158 | 2560 | 2 | 5120
TopMemoryContext | 1 | localtime.cpp | 1965 | 128 | 1 | 128
TopMemoryContext | 1 | localtime.cpp | 1965 | 448 | 1 | 448
(5 rows)
```

pg_shared_chunk_dump(contextname char(64))

Description: This function prints information about all chunks applied for by the memory context in the specified shared memory as a file.

The **contextname** parameter indicates the name of the memory context.

Return type: Boolean

Example:

```
SELECT * FROM pg_shared_chunk_dump('pgstat file hash table');
pg_shared_chunk_dump
-----
t
(1 row)
```

pv_session_chunk_dump(tid int, contextname char(64))

Description: This function prints information about all chunks applied for by a memory context created by a specified thread into a file.

The **tid** parameter indicates the thread ID, while the **contextname** parameter indicates the memory context name.

Return type: Boolean

Example:

```
SELECT * FROM pv_session_chunk_dump(140472797325280, 'Timezones');
pv_session_chunk_dump
-----
t
(1 row)
```

21.7.27.9 Other Functions

pgxc_pool_check()

Description: Checks whether the connection data buffered in the pool is consistent with **pgxc_node**. This parameter has been discarded in 8.3.0 and later cluster versions.

Return type: boolean

pgxc_pool_reload()

Description: Updates the connection information buffered in the pool.

Return type: boolean

pgxc_lock_for_backup()

Description: Locks the cluster before backup. Backup is performed to restore data on new nodes.

Return type: boolean

NOTE

pgxc_lock_for_backup locks a cluster before **gs_dump** or **gs_dumpall** is used to back up the cluster. After a cluster is locked, operations changing the system structure are not allowed. This function does not affect DML statements.

pg_pool_validate(clear boolean, co_node_name cstring)

Description: Clears invalid backend threads on a CN. (These backend threads hold invalid pooler connections to standby DN's.)

Return type: record

pg_nodes_memory()

Description: queries the memory usage of all nodes.

Return type: record

table_skewness(text)

Description: queries the percentage of table data among all nodes.

Parameter: Indicates that the type of the name of the to-be-queried table is text.

Return type: record

table_skewness(table_name text, column_name text[, row_num text])

Description: Queries the proportion of column data distributed on each node based on the hash distribution rule. The results are sorted based on the data volumes of the nodes.

Parameters: **table_name** indicates a table name, **column_name** indicates a column name, and **row_num** indicates that all data in the current column is returned. The default value is **0**. A value other than **0** indicates the number of data records whose statistics are sampled. (Records are randomly sampled.)

Return type: record

Example:

Distribute data by hash based on the **a** column in the **tx** table. Seven records are distributed on DN 1, two records on DN 2, and one record on DN 0.

```
SELECT * FROM table_skewness('tx','a');
seqnum | num | ratio
-----+-----+-----
1      | 7   | 70.000%
2      | 2   | 20.000%
0      | 1   | 10.000%
(3 row)
```

table_data_skewness(data_row record, locatorType "char")

Description: Calculates the bucket distribution index for the records concatenated using the columns in a specified table.

Parameters: **data_row** indicates the record concatenated using columns in the specified table. **locatorType** indicates the distribution rule. You are advised to set **locatorType** to **H**, indicating hash distribution.

Return type: smallint

Example:

Calculates the bucket distribution index based on the hash distribution rule for the records combined concatenated using column **a** in the **tx** table.

```
select a, table_data_skewness(row(a), 'H') from tx;
a | table_data_skewness
--+-----+-----
3 |                    0
6 |                    2
7 |                    2
4 |                    1
5 |                    1
(5 rows)
```

table_distribution(schemaname text, tablename text)

Description: queries the storage space occupied by a specified table on each node.

Parameter: Indicates that the types of the schema name and table name for the table to be queried are both text.

Return type: record

 NOTE

- To query for the storage distribution of a specified table by using this function, you must have the **SELECT** permission for the table.
- The performance of **table_distribution** is better than that of **table_skewness**. Especially in a large cluster with a large amount of data, **table_distribution** is recommended.
- When you use **table_distribution** and want to view the space usage, you can use **dnsize** or **(sum(dnsize) over ())** to view the percentage.

table_distribution(regclass)

Description: queries the storage space occupied by a specified table on each node.

Parameter: indicates the name or OID of the table to be queried. The table name can be defined by the schema name. Parameter type: regclass

Return type: record

 NOTE

- To query for the storage distribution of a specified table by using this function, you must have the **SELECT** permission for the table.
- The performance of **table_distribution** is better than that of **table_skewness**. Especially in a large cluster with a large amount of data, **table_distribution** is recommended.
- When you use **table_distribution** and want to view the space usage, you can use **dnsize** or **(sum(dnsize) over ())** to view the percentage.

table_distribution()

Description: queries the storage distribution of all tables in the current database.

Return type: record

 NOTE

- This function involves the query for information about all tables in the database. To execute this function, you must have the administrator rights or rights of the preset role **gs_role_read_all_stats**.
- Based on the **table_distribution()** function, DWS provides the **PGXC_GET_TABLE_SKEWNESS** view as an alternative way to query for data skew. You are advised to use this view when the number of tables in the database is less than 10000.

gs_table_distribution(schemaname text, tablename text)

Description: queries the storage space occupied by a specified table on each node.

Return type: record

Table 21-58 Fields returned by the **gs_table_distribution(schemaname text, tablename text)** function

| Name | Type | Description |
|------------|------|-------------|
| schemaname | name | Schema name |

| Name | Type | Description |
|----------------|-----------|--|
| tablename | name | Table name |
| relkind | character | Type. <ul style="list-style-type: none">• i: index• r: table |
| relpersistence | character | Type. <ul style="list-style-type: none">• t: local temporary table• g: global temporary table• u: unlogged table• p: common table |
| nodename | name | Node name |
| dnsize | bigint | Storage space of the table on the node, in bytes. |
| sessionid | bigint | For a global temporary table, the session ID of each session is displayed. |

 NOTE

- To query for the storage distribution of a specified table by using this function, you must have the **SELECT** permission for the table.
- This function is based on the physical file storage space records in the **PG_RELFILENODE_SIZE** system catalog. Ensure that the GUC parameters **use_workload_manager** and **enable_perm_space** are enabled.
- The **gs_table_distribution** function performs slower than the **table_distribution** function when querying a single table. However, when querying the entire database, the **gs_table_distribution** function shows significantly better performance. For large clusters with a substantial amount of data, it is recommended to use the **gs_table_distribution** function to query all tables in the database.
- For a global temporary table, the distribution of each session in the storage space is displayed. The **sessionid** column is used to distinguish sessions.

gs_table_distribution()

Description: quickly queries the storage distribution of all tables in the current database.

Return type: record

Table 21-59 Fields returned by the **gs_table_distribution()** function

| Name | Type | Description |
|------------|------|-------------|
| schemaname | name | Schema name |
| tablename | name | Table name |

| Name | Type | Description |
|----------------|-----------|--|
| relkind | character | Type of the table. i : index; r : table. |
| relpersistence | character | Type. <ul style="list-style-type: none">• t: local temporary table• g: global temporary table• u: unlogged table• p: common table |
| nodename | name | Node name |
| dnsize | bigint | Storage space of the table on the node, in bytes. |
| sessionid | bigint | For a global temporary table, the session ID of each session is displayed. |

 NOTE

- To query for the storage distribution of a specified table by using this function, you must have the **SELECT** permission for the table.
- This function is based on the physical file storage space records in the **PG_RELFILENODE_SIZE** system catalog. Ensure that the GUC parameters **use_workload_manager** and **enable_perm_space** are enabled.
- The **gs_table_distribution** function performs slower than the **table_distribution** function when querying a single table. However, when querying the entire database, the **gs_table_distribution** function shows significantly better performance. For large clusters with a substantial amount of data, it is recommended to use the **gs_table_distribution** function to query all tables in the database.
- Based on the **gs_table_distribution()** function, DWS 8.2.1 and later versions provide the **PGXC_WLM_TABLE_DISTRIBUTION_SKEWNESS** view for data skew query. You are advised to use this view when the number of tables in the database is small (less than 10,000).

check_foreign_key_constraint(schema.table)

Description: Checks whether data in a foreign key table meets foreign key constraints. This is supported only by clusters of version 9.1.0.200 or later.

Return type: text

Example:

```
set info_constraint_options = 'foreign_key';

CREATE TABLE ptt1 (a int,b int,e varchar(50),f varchar(50),PRIMARY KEY (a,b)) DISTRIBUTE BY HASH(a,b);
CREATE TABLE tt2 (a int , b int, c int, d int) DISTRIBUTE BY HASH(a,b);

select check_foreign_key_constraint('table_constraints.tt2');
check_foreign_key_constraint
-----
tt2 table has no foreign key constraint

ALTER TABLE tt2 ADD CONSTRAINT fk_tt1_a FOREIGN KEY (a,b) REFERENCES ptt1(a,b);
```

```
select check_foreign_key_constraint('table_constraints.tt2');
check_foreign_key_constraint
-----
tt2 satisfied all Foreign key constraints
```

plan_seed()

Description: Obtains the seed value of the previous query statement (internal use).

Return type: integer

pg_stat_get_env()

Description: Obtains the environment variable information about the current node.

Return type: record

pg_stat_get_thread()

Description: Provides information about the status of all threads under the current node.

Return type: record

pgxc_get_os_threads()

Description: Provides information about the status of threads under all normal nodes in a cluster.

Return type: record

pg_stat_get_sql_count()

Description: Provides statistics on the number of **SELECT/UPDATE/INSERT/DELETE/MERGE INTO** statements executed by all users on the current node, response time, and the number of DDL, DML, and DCL statements.

Return type: record

pgxc_get_sql_count()

Description: Provides statistics on the number of **SELECT/UPDATE/INSERT/DELETE/MERGE INTO** statements executed by all users on all nodes of the current cluster, response time, and the number of DDL, DML, and DCL statements.

Return type: record

pgxc_get_workload_sql_count()

Description: Provides statistics on the number of **SELECT/UPDATE/INSERT/DELETE** statements executed in all workload Cgroup on all CNs of the current cluster and the number of DDL, DML, and DCL statements.

Return type: record

pgxc_get_workload_sql_elapse_time()

Description: Provides statistics on response time of **SELECT/UPDATE/INSERT/DELETE** statements executed in all workload Cgroup on all CNs of the current cluster.

Return type: record

get_instr_unique_sql()

Description: Provides information about Unique SQL statistics collected on the current node. If the node is a CN, the system returns the complete information about the Unique SQL statistics collected on the CN. That is, the system collects and summarizes the information about the Unique SQL statistics on other CNs and DNs. If the node is a DN, the Unique SQL statistics on the DN is returned. For details, see the **GS_INSTR_UNIQUE_SQL** view.

Return type: record

reset_instr_unique_sql(cstring, cstring, INT8)

Description: Clears collected Unique SQL statistics. The input parameters are described as follows:

- **GLOBAL/LOCAL**: Data is cleared from all nodes or the current node.
- **ALL/BY_USERID/BY_CNID/BY_GUC**: **ALL** indicates that all data is cleared. **BY_USERID/BY_CNID** indicates that data is cleared by **USERID** or **CNID**. **BY_GUC** indicates that the clearance operation is caused by the decrease of the value of the GUC parameter **instr_unique_sql_count**.
- The third parameter corresponds to the second parameter. The parameter is invalid for **ALL/BY_GUC**.

Return type: bool

pgxc_get_instr_unique_sql()

Description: Provides complete information about Unique SQL statistics collected on all CNs in a cluster. This function can be executed only on CNs.

Return type: record

get_instr_unique_sql_remote_cns()

Description: Provides complete information about Unique SQL statements collected on all CNs in the cluster, except the CN on which the function is being executed. This function can be executed only on CNs.

Return type: record

pgxc_get_node_env()

Description: Provides the environment variable information about all nodes in a cluster.

Return type: record

gs_switch_relfilenode()

Description: Exchanges meta information of two tables or partitions. (This is only used for the redistribution tool. An error message is displayed when the function is directly used by users).

Return type: integer

copy_error_log_create()

Description: Creates the error table (**public.pgxc_copy_error_log**) required for creating the **COPY FROM** error tolerance mechanism.

Return type: boolean

NOTE

- This function attempts to create the **public.pgxc_copy_error_log** table. For details about the table, see [Table 21-60](#).
- Create the B-tree index on the **relname** column and execute **REVOKE ALL on public.pgxc_copy_error_log FROM public** to manage permissions for the error table (the permissions are the same as those of the **COPY** statement).
- **public.pgxc_copy_error_log** is a row-store table. Therefore, this function can be executed and **COPY FROM** error tolerance is available only when row-store tables can be created in the cluster. After the GUC parameter **enable_hadoop_env** is enabled, row-based tables cannot be created in the cluster. The default value for DWS is **off**.
- Same as the error table and the **COPY** statement, the function requires **sysadmin** or higher permissions.
- If the **public.pgxc_copy_error_log** table or the **copy_error_log_relname_idx** index already exists before the function creates it, the function will report an error and roll back.

Table 21-60 Error table public.pgxc_copy_error_log

| Column | Type | Description |
|-----------|--------------------------|---|
| relname | varchar | Table name in the form of <i>Schema name.Table name</i> |
| begintime | timestamp with time zone | Time when a data format error was reported |
| filename | varchar | Name of the source data file where a data format error occurs |
| rownum | bigint | Number of the row where a data format error occurs in a source data file |
| rawrecord | text | Raw record of a data format error in the source data file To prevent a field from being too long, the length of the field cannot exceed 1024 bytes. |
| detail | text | Error details |

| Column | Type | Description |
|------------|---------|---|
| columnname | varchar | Name of the column whose data format is incorrect in the data source file. Only 8.2.1.100 and later versions support this function. |
| errcode | varchar | Error code corresponding to the error information. The sqlstate error code is used. Only 8.2.1.100 and later versions support this function. |
| queryid | bigint | ID of the SQL statement for executing the Copy statement. It uniquely identifies an SQL statement. Only 8.2.1.100 and later versions support this function. |

pv_compute_pool_workload()

Description: Provides the current load information about computing Node Groups on cloud.

Return type: record

pg_stat_get_status(tid, num_node_display)

Description: Queries for the blocking and waiting status of the backend threads and auxiliary threads in the current instance. For details about the returned results, see the PG_THREAD_WAIT_STATUS view. The input parameters are described as follows:

- **tid**: thread ID, which is of the bigint type. If this parameter is null, the waiting statuses of all backend threads and auxiliary threads are returned. Otherwise, only the waiting statuses of threads with the specified IDs are returned.
- **num_node_display**: integer type. Specifies the maximum number of waiting nodes displayed in the **wait_status** column for records whose waiting status is **wait node**.
 - If this parameter is left empty or set to a value less than or equal to **0**, only one waiting node is displayed.
 - If the value is greater than **20**, a maximum number of nodes can be displayed is **20**.
 - If the value is greater than **0** and less than or equal to **20**, the smaller value between **num_node_display** and the actual number of waiting nodes is displayed. Use the **SELECT * from pg_stat_get_status(NULL, 10)** query for example. If the number of waiting nodes is greater than **10**, the names of only 10 nodes are displayed randomly. If the number of waiting nodes is less than or equal to **10**, the names of all waiting nodes are displayed. If the number of waiting nodes is greater than the number of displayed nodes, the displayed node names are randomly selected.

Return type: record

pgxc_get_thread_wait_status(num_node_display)

Description: Queries for the call hierarchy between threads generated by all SQL statements on each node in a cluster, as well as the block waiting status of each thread. For details about the returned results, see the PGXC_THREAD_WAIT_STATUS view. The type and meaning of the input parameter **num_node_display** are the same as those of the **pg_stat_get_status** function.

Return type: record

pgxc_os_run_info()

Description: Obtains the running status of the operating system on each node in a cluster. For details about the returned results, see "System Catalogs > System Views > PV_OS_RUN_INFO" in the *Developer Guide*.

Return type: record

get_instr_wait_event()

Description: obtains the waiting status and events of the current instance. For details about the returned results, see "System Catalogs > System Views > GS_WAIT_EVENTS" in the *Developer Guide*. If the GUC parameter **enable_track_wait_event** is **off**, this function returns **0**.

Return type: record

pgxc_wait_events()

Description: queries statistics about waiting status and events on each node in a cluster. For details about the returned results, see "System Catalogs > System Views > PGXC_WAIT_EVENTS" in the *Developer Guide*. If the GUC parameter **enable_track_wait_event** is **off**, this function returns **0**.

Return type: record

pgxc_stat_bgwriter()

Description: queries statistics about backend write processes on each node in a cluster. For details about the returned results, see "System Catalogs > System Views > PG_STAT_BGWRITER" in the *Developer Guide*.

Return type: record

pgxc_stat_replication()

Description: queries information about the log synchronization status on each node in a cluster, such as the location where the logs are sent and received. For details about the returned results, see "System Catalogs > System Views > PG_STAT_REPLICATION" in the *Developer Guide*.

Return type: record

pgxc_replication_slots()

Description: queries the replication status on each DN in a cluster. For details about the returned results, see "System Catalogs > System Views > PG_REPLICATION_SLOTS" in the *Developer Guide*.

Return type: record

pgxc_settings()

Description: queries information about runtime parameters on each node in a cluster. For details about the returned results, see "System Catalogs > System Views > PG_SETTINGS" in the *Developer Guide*.

Return type: record

pgxc_instance_time()

Description: queries the running time statistics of each node in a cluster and the time consumed in each execution phase. For details about the returned results, see "System Catalogs > System Views > PV_INSTANCE_TIME" in the *Developer Guide*.

Return type: record

pg_stat_get_redo_stat()

Description: queries Xlog redo statistics on the current node. For details about the returned results, see "System Catalogs > System Views > PV_REDO_STAT" in the *Developer Guide*.

Return type: record

pgxc_redo_stat()

Description: queries the Xlog redo statistics of each node in a cluster. For details about the returned results, see "System Catalogs > System Views > PV_REDO_STAT" in the *Developer Guide*.

Return type: record

get_local_rel_iostat()

Description: Obtains the disk I/O statistics of the current instance. For details about the returned results, see "System Catalogs > System Views > GS_REL_IOSTAT" in the *Developer Guide*.

Return type: record

pgxc_rel_iostat()

Description: queries the disk I/O statistics on each node in a cluster. For details about the returned result, see "System Catalogs > System Views > GS_REL_IOSTAT" in the *Developer Guide*.

Return type: record

get_node_stat_reset_time()

Description: Obtains the time when statistics of the current instance were reset.

Return type: timestampz

pgxc_node_stat_reset_time()

Description: queries the time when the statistics of each node in a cluster are reset. For details about the returned result, see "System Catalogs > System Views > GS_NODE_STAT_RESET_TIME" in the *Developer Guide*.

Return type: record

NOTE

When an instance is running, its statistics keep rising. In the following cases, the statistical values in the memory will be reset to 0:

- The instance is restarted or a cluster switchover occurs.
- The database is dropped.
- A reset operation is performed. For example, the statistics counter in the database is reset using the **pgstat_recv_resetcounter** function or the Unique SQL statements are cleared using the **reset_instr_unique_sql** function.

If any of the preceding events occurs, DWS will record the time when the statistics are reset. You can query the time using the **get_node_stat_reset_time** function.

pgxc_parallel_query(text, text)

Description: Runs a specified SQL query statement on a data instance of a specified type and returns the query result to the current CN. This function is supported in 8.1.2 or later.

The function has two parameters:

The first parameter specifies the instances on which the SQL statement is executed. Currently, the valid input parameters are **dn**, **datanode**, **cn**, **coordinator**, and **all**. Other values will cause function execution errors. **dn** and **datanode** indicate that the statement is executed on all DNs. **cn** and **coordinator** indicate that the statement is executed on all CNs. **all** indicates that the statement is executed on all CNs and DNs.

The second parameter specifies the verification of the objects queried by the SQL statement that is to be sent to a remote node for execution. User tables, distributed tables, and user-defined functions with multiple result sets are not supported.

Return type: record

 NOTE

- This function is only used by developers to efficiently collect the execution information or status of instances in a cluster. You are not advised to use it directly.
- This function contains multiple result sets, and the return data type is record. Therefore, you need to add the output column name and data type specified by the **AS** statement after the function call, as shown in the following:

```
SELECT * FROM pgxc_parallel_query('all', 'select node_name, db_name, thread_name, query_id, tid, lwtid, ptid, tlevel, smpid, wait_status, wait_event from pg_thread_wait_status') AS (node_name text, db_name text, thread_name text, query_id bigint, tid bigint, lwtid integer, ptid integer, tlevel integer, smpid integer, wait_status text, wait_event text);
```
- The data type of the output result of the SQL statement specified by the second parameter of the function must be the same as the data type specified by the **AS** statement. Otherwise, an error may be reported during execution due to type mismatch.
- The SQL statement specified by the second parameter of the function cannot trigger cross-node query. Otherwise, an error is reported.
- The SQL statement specified by the second parameter of the function can only be a **SELECT**, **UPDATE**, **DELETE**, or **INSERT** statement.
 - The **returning** statement is not supported.
 - The user who invokes the function must have the operation permission on the SQL objects.
 - For **INSERT** statements, **INSERT OVERWRITE**, **UPSERT**, and **INSERT INTO** are not supported.
 - The **UPDATE**, **DELETE**, and **INSERT** statements can be executed only by the initial user in in-place upgrade mode or by the administrator in redistribution mode. The number of records modified by the statements on each instance must be the same. Otherwise, an error will be reported during statement execution. The function outputs a column of values of the bigint type. These values indicate the number of records operated by the statement on each instance.

```
SELECT * FROM pgxc_parallel_query('cn', 'UPDATE pg_partition SET relpages = 0') AS (updated bigint);
```

create_wdr_snapshot()

Description: Creates a performance data snapshot.

Return type: text

 NOTE

- Only the database administrator **SYSADMIN** can execute this function.
- This function can be executed only on CNs. If it is executed on DN, the following message will be returned: "WDR snapshot can only be created on coordinator."
- Before executing this function, ensure that the value of **enable_wdr_snapshot** is **on**. If its value is **off**, the following message will be returned for this function: "WDR snapshot request cannot be executed, because GUC parameter 'enable_wdr_snapshot' is off."
- If the snapshot thread is not started for some reason, for example, the node is restarted, the following message will be returned for this function: "WDR snapshot request cannot be accepted, please retry later."
- If this function fails to be executed, the following message will be returned: "Cannot respond to WDR snapshot request."
- If this function is successfully executed, the following message will be returned: "WDR snapshot request has been submitted." This message indicates that the snapshot creation request has been sent to the background snapshot thread, but does not mean that the snapshot has been successfully created.

kill_snapshot(scope cstring)

Description: kills the background snapshot thread. This function sends a command to the background snapshot thread and waits for the thread to stop.

The input parameter **scope** indicates the operation scope. Its value can be **local** or **global**.

- Value **local** indicates killing the snapshot thread on the current CN.
- Value **global** indicates killing the snapshot thread on the current CN as well as those on all the other CNs in the cluster.
- If any other value is passed, error message "Scope is invalid, use "local" or "global"." is displayed.
- The input parameter can be left empty, in which case the default value **local** will be used.

Return type: none

NOTE

- Only the database administrator **SYSADMIN** can execute this function.
- This function can be executed only on CNs. If it is executed on DN, the following message will be returned: "kill_snapshot can only be executed on coordinator."
- Executing this function sends a kill signal to the background snapshot thread and waits for it to finish. If the snapshot thread is not killed within 100s, the error message "Kill snapshot thread failed" is displayed.

generate_wdr_report(begin_snap_id bigint, end_snap_id bigint, report_type cstring, report_scope cstring, node_name cstring)

Description: Creates a load analysis report.

The input parameters are described as follows:

- **begin_snap_id** and **end_snap_id**: IDs of the start and end snapshots, respectively. The IDs are of the bigint type. The value of **begin_snap_id** must be less than that of **end_snap_id**, and the time for the start and end snapshots cannot overlap. You can check whether the snapshot time overlaps by querying **select s1.end_ts < s2.start_ts from (select * from dbms_om.snapshot where snapshot_id=end_snap_id) as s2;** in the **dbms_om.snapshot** table. If **true** is returned, the snapshot time does not overlap. Otherwise, the snapshot time overlaps.
- **report_type**: report type. The value is a cstring and can be **summary**, **detail**, or **all**.
- **report_scope**: report scope. The value is a cstring and can be **cluster** or **node**.
- **node_name**: node name. The value is a cstring. If **report_scope** is **node**, the value of this parameter must be **pg_catalog**, which indicates the CN or DN name in the **node_name** column of the **pgxc_node** table.

Return type: text

 NOTE

- Only the database administrator **SYSADMIN** can execute this function.
- This function can be executed only on CNs. If it is executed on DN, the following message will be returned: "WDR report can only be created on coordinator."
- If the report is created successfully, message "Report %s has been generated" will be returned.
- The statistics cannot be reset between the time the start snapshot is taken and the time the end snapshot is taken. Otherwise, error message "Instance reset time is different" will be displayed. For details about the events that cause a statistics reset, see the [pgxc_node_stat_reset_time](#) function.

wdr_xdb_query(db_name text, snapshot_id bigint, view_name text)

Description: Queries a specified view in a specified database. The query results of some views vary depending on databases. For example, the **global_table_stat** view is used to query the statistics of a table. The results of querying this view vary because tables in different databases are different. The **wdr_xdb_query** function can access the database specified by **db_name** in the current connection and query the view specified by **view_name** in the database. The input parameters are described as follows:

- **db_name**: specifies the name of a database. The value is of the text type.
- **snapshot_id**: specifies the snapshot ID. The value is of the bigint type. For details, see section "Performance View Snapshot" in *Data Warehouse Service (DWS) Developer Guide*.
- **view_name**: specifies the name of a view. The value is of the text type. The view name must be in the following whitelist:
 - global_table_stat
 - global_table_change_stat
 - global_column_table_io_stat
 - global_row_table_io_stat

The return value type is record. The first column is **snapshot_id bigint**, and the second column is **db_name text**. The names, types, and sequences of other columns are the same as those of the views specified by **view_name**.

Example:

```
select snapshot_id, db_name, schemaname, relname, distribute_mode,
seq_scan ,seq_tuple_read ,index_scan ,index_tuple_read ,tuple_inserted
,tuple_updated ,tuple_deleted ,tuple_hot_updated ,live_tuples ,dead_tuples from
wdr_xdb_query('postgres':text, 1, 'global_table_stat':text) as i(snapshot_id bigint, db_name text,
schemaname name, relname name, distribute_mode char, seq_scan bigint, seq_tuple_read
bigint, index_scan bigint, index_tuple_read bigint, tuple_inserted bigint, tuple_updated bigint,
tuple_deleted bigint, tuple_hot_updated bigint, live_tuples bigint, dead_tuples bigint);
```

 NOTE

- This function is supported only in 8.1.2 or later.
- Only the database administrator **SYSADMIN** can execute this function.
- This function can be used to query only the views in the whitelist. If you use this function to query other views, the error message **Input view name is invalid.** will be displayed.

vac_fileclear_relation(oid)

Description: Forcibly clears VACUUM rewritten files in a specified column-store table to reclaim space.

Parameter: OID of a column-store table.

Return type: integer

NOTE

- Before using this function, set **colvacuum_threshold_scale_factor** and ensure that the files are cleared and space reclaimed only after the VACUUM process has rewritten the files of the specified column-store table.
- This function exclusively locks a specified column-store table.
- This function cannot be used after DDL delay is enabled.

vac_fileclear_all_relation()

Description: Forcibly clears VACUUM rewritten files in all specified column-store tables to reclaim space.

Return type: record

NOTE

This function cannot be used after DDL delay is enabled.

get_col_file_info(table_name)

Description: Queries the number of empty CU files and the total number of CU files in a specified column-store table. This function is supported only in cluster 8.2.0 and later versions.

Parameter: Name of a column-store table.

Return type: int

Fields in the returned value:

- **total_file_num int**: total number of CU files. The value ranges from **-1** to **INT_MAX**. **-1** indicates a failure, which could be caused by unsupported table types. The values in the range **0** to **INT_MAX** indicates the total number of files.
- **empty_file_num int**: number of empty CU files. The value ranges from **-1** to **INT_MAX**. **-1** indicates a failure, which could be caused by unsupported table types. The values in the range **0** to **INT_MAX** indicates the total number of empty files.

Example:

```
call get_col_file_info('t4');
total_file_num | empty_file_num
-----+-----
10 | 7
(1 row)
```

get_all_col_file_info()

Description: Queries the number of empty CU files and the total number of CU files in all column-store tables. This function is supported only in cluster 8.2.0 and later versions.

Return type: record

Fields in the returned value:

- **space_name text**: schema to which the column-store tables belong
- **table_name text**: name of a column-store table
- **total_file_num int**: total number of CU files. The value ranges from **-1** to **INT_MAX**. **-1** indicates a failure, which could be caused by unsupported table types. The values in the range **0** to **INT_MAX** indicates the total number of files.
- **empty_file_num int**: number of empty CU files. The value ranges from **-1** to **INT_MAX**. **-1** indicates a failure, which could be caused by unsupported table types. The values in the range **0** to **INT_MAX** indicates the total number of empty files.

Example:

```
call get_all_col_file_info();
space_name | table_name | total_file_num | empty_file_num
-----+-----+-----+-----
public    | t4        | 10             | 7
public    | t2        | 1              | 1
public    | t1        | 3              | 0
(3 rows)
```

pg_get_bucket_epoch(relid)

Description: Queries the **epoch** value of the **v3** table on the current node. To print the **epoch** values of all subpartitions in a partitioned table, you must provide the OID of the parent table. This function can be executed only in VW write scenarios. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: record

Fields in the returned value:

- **relfilenode oid**: column-store **relfilenode** of a non-partitioned table. For a partitioned table, the value is **0**.
- **partfilenode oid**: **relfilenode** of a column-store partitioned table. For a non-partitioned table, the value is **0**.
- **epoch xid**: **epoch** value corresponding to the table on the current node.

Example:

```
select * from pg_get_bucket_epoch(2147483732);
relfilenode | partfilenode | epoch
-----+-----+-----
2137383742 | 0            | 1000001
(1 row)
```

pg_get_bucket_epoch(tablespaceid, databaseid, relfilenode)

Description: Queries the **epoch** value of the **v3** table on the current node. Enter the OID of the tablespace where the **v3** table is located, the OID of the database,

and the **relfilenode** corresponding to the table. This function can be executed in both VW write and read scenarios. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: xid

Fields in the returned value:

epoch xid: **epoch** value corresponding to the table on the current node.

Example:

```
select * from pg_get_bucket_epoch(2147483729, 16792, 2147483732);
epoch
-----
1000001
(1 row)
```

pgxc_get_bucket_epoch(tablespaceid, databaseid, relfilenode)

Description: Queries the **epoch** values of the **v3** table on all DN. Enter the OID of the tablespace where the **v3** table is located, the OID of the database, and the **relfilenode** corresponding to the table. This function can be executed in both VW write and read scenarios. This function is supported only by clusters of version 9.1.0.200 or later.

Return type: record

Fields in the returned value:

- **nodename text:** node name
- **epoch xid:** **epoch** value corresponding to the table on the current node.

Example:

```
select * from pgxc_get_bucket_epoch(2147483729, 16792, 2147483732);
nodename | epoch
-----+-----
datanode3 | 1000001
datanode1 |      0
datanode2 |      0
(1 row)
```

current_temp_schema()

Description: Returns the temporary schema of the current session. If the current session does not have a temporary schema, null is returned. This function is supported by clusters of version 8.2.1.220 or later.

Return type: name

Example:

```
select current_temp_schema();
current_temp_schema
-----
(1 row)
create temp table tmp1(a int,b int) distribute by hash(a);
CREATE TABLE
select current_temp_schema();
current_temp_schema
-----
```

```
pg_temp_coordinator1_2_2_140718823485088
(1 row)
```

get_volatile_pg_class()

Description: Obtains the **pg_class** metadata related to all volatile temporary tables in the current session. This parameter is supported by version 8.2.0 or later clusters.

Return type: record

Fields in the returned value:

- **oid**: OID of the volatile temporary table.
- Other fields: same as the fields (excluding hidden fields) in the **pg_class system catalog**.

get_volatile_pg_class(relname text)

Description: Obtains the **pg_class** metadata related to a specified volatile temporary table in the current session. This parameter is supported by version 8.2.0 or later clusters.

Parameter: name of the **volatile** temporary table in the current session.

Return type: record

Fields in the returned value:

- **oid**: OID of the volatile temporary table.
- Other fields: same as the fields (excluding hidden fields) in the **pg_class system catalog**.

Example:

```
SELECT * FROM get_volatile_pg_class('tx1');
 oid | relname | relnamespace | reltype | reloftype | relowner | relam | relfilenode | reltablespace | relpages |
 reltuples | relallvisible | reltoastrelid | reltoastidxid | reldeltarelid |
 reldeltaidx | relcudescrid | relcudescidx | relhasindex | relisshared | relpersistence | relkind | relnatts |
 relchecks | relhasoids | relhaspkey | relhasrules | relhastriggers | relhassub
 class | relcmprr | relhasclusterkey | relrowmovement | parttype | relfrozenxid | relacl |
 reloptions | relreplident | relfrozenxid64
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
16772 | tx1 | 16770 | 16774 | 0 | 10 | 0 | 16772 | 1665 | 0 | 0
| 0 | 16775 | 0 | 0 |
| 0 | 0 | 0 | f | f | v | r | 2 | 0 | f | f |
f | f | f | f | n | 11815 | | {orientation=row,compression=no} |
d | 11815
(1 row)
```

get_volatile_pg_attribute()

Description: Obtains the **pg_attribute** metadata related to all volatile temporary tables in the current session. This parameter is supported by version 8.2.0 or later clusters.

Return type: record

Fields in the returned value:

- **oid**: OID of the column.
- Other fields: same as the fields (excluding hidden fields) in the **pg_attribute** catalog.

get_volatile_pg_attribute(relname text, attrname text)

Description: Obtains the **pg_attribute** metadata related to a specified volatile temporary table in the current session. This parameter is supported by version 8.2.0 or later clusters.

Parameter:

- **relname**: table name (must be in the current session).
- **attrname**: column name.

Return type: record

Fields in the returned value:

- **oid**: OID of the column.
- Other fields: same as the fields (excluding hidden fields) in the **pg_attribute** catalog.

Example:

```
SELECT * FROM get_volatile_pg_attribute('tx1', 'b');
 attrelid | attname | atttypid | attstattarget | attlen | attnum | attndims | attcacheoff | atttypmod | attbyval |
 attstorage | attalign | attnotnull | atthasdef | attisdropped | attislocal |
 attcmprmode | attinhcount | attcollation | attacl | attoptions | attfdwoptions | attinitdefval | attkvtype
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 16772 | b       | f         |      25       |    -1   |     -1   |     2     |     0        |    -1     |    -1     | f         | x         | i         |
 127   |         |           |      0        |    100  |          |           |             |           |           | 0         |           |           |
(1 row)
```

pg_get_publication_tables(pubname text)

Description: Returns the relid list of tables to be published based on the publication name. This function is supported by version 8.2.0.100 or later clusters.

Parameter: **pubname**

Return type: set of OID

Example:

```
SELECT * FROM pg_get_publication_tables('mypub');
 relid
-----
 16757
 16776
(2 rows)
```

pg_relation_is_publishable(relname regclass)

Description: Checks whether a table can be published. This function is supported by version 8.2.0.100 or later clusters.

Parameter: **relname**

Return type: Boolean

Example:

```
SELECT * FROM pg_relation_is_publishable('t1');
pg_relation_is_publishable
-----
t
(1 row)
```

get_col_cu_info(schema_name text, table_name text, row_count int8, dirty_percent int8)

Description: Queries the CU information of a column-store table. The CU information of each partition is collected separately. This function is supported by version 8.2.0.100 or later clusters.

Parameters: schema name (mandatory), table name (mandatory), threshold for the number of rows in a small CU (optional, 200 by default, ranging from 1 to 60000, and percentage threshold for deleting dirty CUs (optional, 70 by default, ranging from 1 to 100)

Return type: record

Fields in the returned value:

node_name: DN name.

part_name: partition name. This column is empty for a common table.

zero_size_cu_count: number of CUs whose **cuSize** is **0** and number of rows is less than or equal to **row_count**.

small_cu_count: number of CUs whose **cuSize** is **ALIGNOF_CUSIZE(8192)** and number of rows is less than or equal to **row_count**.

dirty_cu_count: number of CUs whose deadtupe percentage exceeds **dirty_percent** due to deletion.

total_cu_count: total number of CUs.

small_cu_size: total size of 8 KB CUs.

total_cu_size: total CU size.

Example:

```
SELECT * FROM get_col_cu_info('public','hs_part');
node_name | part_name | zero_size_cu_count | small_cu_count | dirty_cu_count | total_cu_count |
small_cu_size | total_cu_size
-----+-----+-----+-----+-----+-----+-----+-----
dn_1      | p1        | 3 | 0 | 0 | 3 | 0 bytes | 0 bytes
dn_1      | p2        | 3 | 0 | 0 | 3 | 0 bytes | 0 bytes
dn_1      | p3        | 3 | 0 | 0 | 3 | 0 bytes | 0 bytes
(3 rows)
```

```
SELECT * FROM get_col_cu_info('public','hs_part', 200, 90);
node_name | part_name | zero_size_cu_count | small_cu_count | dirty_cu_count | total_cu_count |
small_cu_size | total_cu_size
-----+-----+-----+-----+-----+-----+-----+-----
dn_1      | p1        |          3         |          0      |          0      |          3      | 0 bytes | 0 bytes
dn_1      | p2        |          3         |          0      |          0      |          3      | 0 bytes | 0 bytes
dn_1      | p3        |          3         |          0      |          0      |          3      | 0 bytes | 0 bytes
(3 rows)
```

get_col_file_vacuum_info(schema_name text, table_name text, force_get_rewritten_file_num bool)

Description: Queries the vacuum information of a column-store table. The vacuum information of each partition is collected separately. This function is supported by version 8.2.0.100 or later clusters.

Parameters: schema name (mandatory), table name (mandatory), and whether to forcibly obtain the precise number of files that can be cleared (mandatory, **false** by default)

Return type: record

Fields in the returned value:

node_name: DN name.

part_name: partition name. This column is empty for a common table.

total_file_num: total number of CU files.

rewritable_file_num: number of files that can be rewritten but have not been rewritten.

rewritten_file_num: number of files that have been rewritten but have not been cleared. The value is obtained from data in the memory. If the memory data is lost due to reasons such as restart, you can set **force_get_rewritten_file_num=true** to forcibly obtain the accurate number of files that can be cleared.

empty_file_num: number of cleared files.

Example:

```
SELECT * FROM get_col_file_vacuum_info('public','pa',false);
node_name | part_name | total_file_num | rewritable_file_num | rewritten_file_num | empty_file_num
-----+-----+-----+-----+-----+-----
datanode1 | pa1       |          1     |          0          |          0          |          0
datanode1 | pa2       |          1     |          0          |          0          |          0
datanode2 | pa1       |          1     |          0          |          0          |          0
datanode2 | pa2       |          1     |          0          |          0          |          0
datanode3 | pa1       |          1     |          0          |          0          |          0
datanode3 | pa2       |          1     |          0          |          0          |          0
(6 rows)
```

get_col_file_vacuum_info(schema_name text, table_name text, colvacuum_threshold_scale_factor int)

Description: Queries the vacuum information of a column-store table. The vacuum information of each partition is collected separately. This function is supported by version 8.2.0.100 or later clusters.

Parameters: schema name (mandatory), table name (mandatory), and **colvacuum_threshold_scale_factor** (mandatory. The value range is 0 to 100, indicating the ratio of dead tuples.)

Return type: record

Return value:

node_name: DN name.

part_name: partition name. This column is empty for a common table.

total_file_num: total number of CU files.

rewritable_file_num: number of files that can be rewritten but have not been rewritten.

rewritten_file_num: number of files that have been rewritten but have not been cleared. The value is obtained from data in the memory. If the memory data is lost due to reasons such as restart, you can set **force_get_rewritten_file_num=true** to forcibly obtain the accurate number of files that can be cleared.

empty_file_num: number of cleared files.

Example:

```
SELECT * FROM get_col_file_vacuum_info('public','pa',10);
```

| node_name | part_name | total_file_num | rewritable_file_num | rewritten_file_num | empty_file_num |
|-----------|-----------|----------------|---------------------|--------------------|----------------|
| datanode1 | pa1 | 1 | 0 | 0 | 0 |
| datanode1 | pa2 | 1 | 0 | 0 | 0 |
| datanode2 | pa1 | 1 | 0 | 0 | 0 |
| datanode2 | pa2 | 1 | 0 | 0 | 0 |
| datanode3 | pa1 | 1 | 0 | 0 | 0 |
| datanode3 | pa2 | 1 | 0 | 0 | 0 |

(6 rows)

get_all_col_cu_info(row_count int8)

Description: Queries the CU information of all column-store tables in the database. This function is supported by version 8.2.0.100 or later clusters.

Parameter: threshold for the number of rows in a small CU (optional, **200** by default, and ranging from **1** to **60000**)

Return type: record

Fields in the returned value:

node_name: DN name.

schema_name: schema name.

table_name: table name.

zero_size_cu_count: number of CUs whose **cuSize** is **0** and number of rows is less than or equal to **row_count**.

small_cu_count: number of CUs whose **cuSize** is **ALIGNOF_CUSIZE(8192)** and number of rows is less than or equal to **row_count**.

total_cu_count: total number of CUs.

small_cu_size: total size of 8 KB CUs.

total_cu_size: total CU size.

Example:

```
SELECT * FROM get_all_col_cu_info(200);
```

| node_name | schema_name | table_name | zero_size_cu_count | small_cu_count | total_cu_count | small_cu_size | total_cu_size |
|-----------|-------------|------------|--------------------|----------------|----------------|---------------|---------------|
| datanode1 | public | udi_48076 | 5 | 1 | 6 | 8192 bytes | 8192 bytes |
| datanode1 | public | udi_48077 | 5 | 1 | 6 | 8192 bytes | 8192 bytes |
| datanode2 | public | udi_48076 | 5 | 1 | 6 | 8192 bytes | 8192 bytes |
| datanode2 | public | udi_48077 | 5 | 1 | 6 | 8192 bytes | 8192 bytes |
| datanode3 | public | udi_48076 | 5 | 1 | 6 | 8192 bytes | 8192 bytes |
| datanode3 | public | udi_48077 | 5 | 1 | 6 | 8192 bytes | 8192 bytes |

(6 rows)

get_all_col_file_vacuum_info(force_get_rewritten_file_num bool)

Description: Queries the vacuum information of all column-store tables in the database. This function is supported by version 8.2.0.100 or later clusters.

Parameter: whether to forcibly obtain the accurate number of files that can be cleared (mandatory. It can be **true** or **false**.)

Return type: record

Fields in the returned value:

node_name: DN name.

schema_name: schema name.

table_name: table name.

total_file_num: total number of CU files.

rewritable_file_num: number of files that can be rewritten but have not been rewritten.

rewritten_file_num: number of files that have been rewritten but have not been cleared. The value is obtained from data in the memory. If the memory data is lost due to reasons such as restart, you can set **force_get_rewritten_file_num=true** to forcibly obtain the accurate number of files that can be cleared.

empty_file_num: number of cleared files.

Example:

```
SELECT * FROM get_all_col_file_vacuum_info(false);
```

| node_name | schema_name | table_name | total_file_num | rewritable_file_num | rewritten_file_num | empty_file_num |
|-----------|-------------|------------|----------------|---------------------|--------------------|----------------|
| datanode1 | public | udi_57373 | 2 | 0 | 0 | 1 |
| datanode1 | public | udi_57374 | 2 | 0 | 0 | 1 |
| datanode2 | public | udi_57373 | 2 | 0 | 0 | 1 |
| datanode2 | public | udi_57374 | 2 | 0 | 0 | 1 |
| datanode3 | public | udi_57373 | 2 | 0 | 0 | 1 |
| datanode3 | public | udi_57374 | 2 | 0 | 0 | 1 |

show_tsc_info()

Description: Queries the TimeStamp-Counter (TSC) information obtained from the current database node. This function is supported by version 8.2.1 or later clusters.

Return type: record

Table 21-61 Parameter

| Name | Type | Description |
|-----------------------|---------|---|
| node_name | text | Node name |
| tsc_mult | bigint | TSC conversion multiplier |
| tsc_shift | bigint | TSC conversion shifts |
| tsc_frequency | float8 | TSC frequency. |
| tsc_use_frequency | boolean | Indicates whether to use the TSC frequency for time conversion. |
| tsc_ready | boolean | Indicates whether the TSC frequency can be used for time conversion |
| tsc_scalar_error_info | text | Error information about obtaining TSC conversion information |
| tsc_freq_error_info | text | Error information about obtaining TSC frequency information |

Example:

```
SELECT * FROM show_tsc_info();
 node_name | tsc_mult | tsc_shift | tsc_frequency | tsc_use_frequency | tsc_ready |
 tsc_scalar_error_info | tsc_freq_error_info
-----+-----+-----+-----+-----+-----+-----
 coordinator1 |          |          | 2400 | t          | t          | TSC scalar is not initialized |
```

get_tsc_info()

Description: Re-obtains the TimeStamp-Counter (TSC) information of the current database node. This function is supported by version 8.2.1 or later clusters.

Return type: record

Table 21-62 show_tsc_info() return columns

| Column | Type | Description |
|-----------|--------|---------------------------|
| node_name | text | Node name |
| tsc_mult | bigint | TSC conversion multiplier |
| tsc_shift | bigint | TSC conversion shifts |

| Column | Type | Description |
|-----------------------|---------|---|
| tsc_frequency | float8 | TSC frequency |
| tsc_use_frequency | boolean | Indicates whether to use the TSC frequency for time conversion. |
| tsc_ready | boolean | Indicates whether the TSC frequency can be used for time conversion |
| tsc_scalar_error_info | text | Error information about obtaining TSC conversion information |
| tsc_freq_error_info | text | Error information about obtaining TSC frequency information |

Example:

```
SELECT * FROM get_tsc_info();
 node_name | tsc_mult | tsc_shift | tsc_frequency | tsc_use_frequency | tsc_ready |
 tsc_scalar_error_info | tsc_freq_error_info
-----+-----+-----+-----+-----+-----+-----
 coordinator1 |          |          | 2400 | t          | t          | TSC scalar is not initialized |
```

test_tsc_info(time float8, loops int)

Description: Tests the accuracy of the time converted using the TimeStamp-Counter (TSC) on the current node. This function is supported by version 8.2.1 or later clusters.

The input parameters are described as follows:

- **time**: indicates the test time difference (unit: s). The test duration must be less than or equal to 60s.
- **loops**: indicates the number of tests. The value ranges from 1 to 10.

Return type: record

Fields in the returned value:

- **id**: number of cycles.
- **real_time_diff**: time difference obtained using **gettimeofday** (unit: us).
- **est_time_scalar**: time difference (unit: μ s) converted using TSC conversion information.
- **est_time_frequency**: time difference (unit: μ s) converted using the TSC frequency.

Example:

```
SELECT * FROM test_tsc_info(0.01,10);
 id | real_time_diff | est_time_scalar | est_time_frequency
-----+-----+-----+-----
 1 | 10057 | 10056.9
 2 | 10057 | 10057.4816666667
 3 | 10056 | 10055.2841666667
 4 | 10054 | 10054.4908333333
```

| | | | | | | |
|----|--|-------|--|--|--|------------------|
| 5 | | 10055 | | | | 10054.2875 |
| 6 | | 10055 | | | | 10054.7483333333 |
| 7 | | 10055 | | | | 10054.4725 |
| 8 | | 10054 | | | | 10054.0766666667 |
| 9 | | 10058 | | | | 10058.1016666667 |
| 10 | | 10057 | | | | 10056.3733333333 |

(10 rows)

pgxc_recyclebin_purge

Description: This function traverses and clears objects in the recycle bin. It is supported only by 9.1.0.200 and later cluster versions.

Return type: void

Example:

```
SELECT * FROM pgxc_recyclebin_purge();
pgxc_recyclebin_purge
-----
(1 row)
```

21.7.28 Backup and Restoration Control Functions

21.7.28.1 Backup Control Functions

Backup control functions help online backup.

pg_create_restore_point(name text)

Description: Creates a named point for performing the restore operation (restricted to system administrators).

Return type: text

Note: **pg_create_restore_point** creates a named transaction log record that can be used as a restoration target, and returns the corresponding transaction log location. The given name can then be used with **recovery_target_name** to specify the point up to which restoration will proceed. Avoid creating multiple restoration points with the same name, since restoration will stop at the first one whose name matches the restoration target.

pg_current_xlog_location()

Description: Obtains the write position of the current transaction log.

Return type: text

Note: **pg_current_xlog_location** displays the write position of the current transaction log in the same format as those of the previous functions. Read-only operations do not require rights of the system administrator.

Example:

```
SELECT pg_current_xlog_location();
pg_current_xlog_location
-----
0/4310D5D8
(1 row)
```

pg_current_xlog_insert_location()

Description: Obtains the insert position of the current transaction log.

Return type: text

Note: **pg_current_xlog_insert_location** displays the insert position of the current transaction log. The insertion point is the logical end of the transaction log at any instant, while the write location is the end of what has been written out from the server's internal buffers. The write position is the end that can be detected externally from the server. This operation can be performed to archive only some of completed transaction log files. The insert position is mainly used for commissioning the server. Read-only operations do not require rights of the system administrator.

Example:

```
SELECT pg_current_xlog_insert_location();
pg_current_xlog_insert_location
-----
0/4316AD50
(1 row)
```

pg_start_backup(label text [, fast boolean])

Description: Starts executing online backup (restricted to system administrators or replication roles).

Return type: text

Note: **pg_start_backup** receives a user-defined backup label (usually the name of the position where the backup dump file is stored). This function writes a backup label file to the data directory of the database cluster and then returns the starting position of backed up transaction logs in text mode.

```
SELECT pg_start_backup('label_goes_here');
pg_start_backup
-----
0/3000020
(1 row)
```

pg_stop_backup()

Description: Completes online backup (restricted to system administrators or replication roles).

Return type: text

Note: **pg_stop_backup** deletes the label file created by **pg_start_backup** and creates a backup history file in the transaction log archive area. The history file includes the label given to **pg_start_backup**, the starting and ending transaction log locations for the backup, and the starting and ending times of the backup. The return value is the backup's ending transaction log location. After the ending position is calculated, the insert position of the current transaction log automatically goes ahead to the next transaction log file. This way, the ended transaction log file can be immediately archived so that backup is complete.

Example:

```
SELECT pg_stop_backup();
pg_stop_backup
```

```
-----  
0/4401E610  
(1 row)
```

pg_switch_xlog()

Description: Switches to a new transaction log file (restricted to system administrators).

Return type: text

Note: **pg_switch_xlog** moves to the next transaction log file so that the current log file can be archived (if continuous archive is used). The return value is the ending transaction log location + 1 within the just-completed transaction log file. If there has been no transaction log activity since the last transaction log switchover, **pg_switch_xlog** will do nothing but return the start location of the transaction log file currently in use.

Example:

```
SELECT pg_switch_xlog();  
pg_switch_xlog  
-----  
0/45017978  
(1 row)
```

pg_xlogfile_name(location text)

Description: Converts the position string in a transaction log to a file name.

Return type: text

Note: **pg_xlogfile_name** extracts only the transaction log file name. If the given transaction log position is the transaction log file border, a transaction log file name will be returned for both the two functions. This is usually the desired behavior for managing transaction log archiving, since the preceding file is the last one that currently needs to be archived.

pg_xlogfile_name_offset(location text)

Description: Converts the position string in a transaction log to a file name and returns the byte offset in the file.

Return type: text, integer

Note: **pg_xlogfile_name_offset** can extract transaction log file names and byte offsets from the returned results of the preceding functions. Example:

```
SELECT * FROM pg_xlogfile_name_offset(pg_stop_backup());  
NOTICE: pg_stop_backup cleanup done, waiting for required WAL segments to be archived  
NOTICE: pg_stop_backup complete, all required WAL segments have been archived  
  file_name      | file_offset  
-----+-----  
000000010000000000000003 |      272  
(1 row)
```

pg_xlog_location_diff(location text, location text)

Description: **pg_xlog_location_diff** calculates the difference in bytes between two transaction log locations.

Return type: numeric

`pg_cbm_tracked_location()`

Description: Queries for the LSN location parsed by CBM.

Return type: text

`pg_cbm_get_merged_file(startLSNArg text, endLSNArg text)`

Description: Combines CBM files within the specified LSN range into one and returns the name of the combined file.

Return type: text

`pg_cbm_get_changed_block(startLSNArg text, endLSNArg text)`

Description: Combines CBM files within the specified LSN range into a table and return records of this table.

Return type: record

Note: The table columns include the start LSN, end LSN, tablespace OID, database OID, table relfilenode, table fork number, whether the table is deleted, whether the table is created, whether the table is truncated, number of pages in the truncated table, number of modified pages, and list of No. of modified pages.

`pg_cbm_recycle_file(slotName name, targetLSNArg text)`

Description: Deletes the CBM files that are no longer used and returns the first LSN after the deletion. If **slotName** is empty, **targetLSNArg** is used as the recycling point. During backup and DR, you need to specify a slot name due to parallelism. Record the **targetLSNArg** value of the task to the slot, traverse all backup slots, and find the smallest LSN as the recycling point.

Return type: text

`pg_cbm_force_track(targetLSNArg text,timeOut int)`

Description: Forcibly executes the CBM trace to the specified Xlog position and returns the Xlog position of the actual trace end point.

Return type: text

`pg_enable_delay_ddl_recycle()`

Description: Enables DDL delay and returns the Xlog position of the enabling point.

Return type: text

Example:

```
SELECT pg_enable_delay_ddl_recycle();
pg_enable_delay_ddl_recycle
-----
00000000/4607B250
(1 row)
```

pg_disable_delay_ddl_recycle(barrierLSNArg text, isForce bool)

Description: Disables DDL delay and returns the Xlog range where DDL delay takes effect.

Return type: record

pg_enable_delay_xlog_recycle()

Description: Enables Xlog recycle delay.

Return type: void

Example:

```
SELECT pg_enable_delay_xlog_recycle();
pg_enable_delay_xlog_recycle
-----
(1 row)
```

pg_disable_delay_xlog_recycle()

Description: Disables Xlog recycle delay.

Return type: void

Example:

```
SELECT pg_disable_delay_xlog_recycle();
pg_disable_delay_xlog_recycle
-----
(1 row)
```

pgxc_get_senders_catchup_time()

Description: Displays the catchup information of the currently active primary/standby instance sending thread on all DN's.

Return type: record

The following information is returned:

Table 21-63 pgxc_get_senders_catchup_time() columns

| Column | Type | Description |
|------------|---------|-------------------------------------|
| node_name | text | Node name |
| lwpid | integer | Current sender lwpid |
| local_role | text | Local role |
| peer_role | text | Peer role |
| state | text | Current sender's replication status |
| sender | text | Current sender type |

| Column | Type | Description |
|------------------------|--------------------------|--|
| catchup_start | timestamp with time zone | Startup time of a catchup task |
| catchup_end | timestamp with time zone | End time of a catchup task |
| catchup_type | text | Catchup task type, full or incremental |
| catchup_bcm_filename | text | BCM file executed by the current catchup task |
| catchup_bcm_finished | integer | Number of BCM files completed by a catchup task |
| catchup_bcm_total | integer | Total number of BCM files to be operated by a catchup task |
| catchup_percent | text | Completion percentage of a catchup task |
| catchup_remaining_time | text | Estimated remaining time of a catchup task |

pgxc_lock_for_backup()

Description: Locks the cluster before backup. Backup is performed to restore data on new nodes.

Return type: boolean

NOTE

pgxc_lock_for_backup locks a cluster before **gs_dump** or **gs_dumpall** is used to back up the cluster. After a cluster is locked, operations changing the system structure are not allowed. This function does not affect DML statements.

Example:

```
SELECT pgxc_lock_for_backup();
pgxc_lock_for_backup
-----
t
(1 row)
```

21.7.28.2 Restoration Control Functions

Restoration control functions provide information about the status of standby nodes. These functions can be executed during restoration or when the database is running properly.

pg_is_in_recovery()

Description: Returns **true** if restoration is still in progress.

Return type: bool

Example:

```
SELECT pg_is_in_recovery();
pg_is_in_recovery
-----
f
(1 row)
```

pg_last_xlog_receive_location()

Description: This function gets the last transaction log location received and synchronized to disk by streaming replication. While streaming replication is in progress, this will increase monotonically. If restoration has completed, then this value will remain static at the value of the last WAL record received and synchronized to disk during restoration. If streaming replication is disabled or if not yet started, the function return will return **NULL**.

Return type: text

Example:

```
SELECT pg_last_xlog_receive_location();
pg_last_xlog_receive_location
-----
(1 row)
```

pg_last_xlog_replay_location()

Description: Gets last transaction log location replayed during restoration. When the recovery is still in progress, the transaction log keeps increasing. If restoration has completed, then this value will remain static at the value of the last WAL record received during that restoration. When the server has been started normally without restoration, the function returns **NULL**.

Return type: text

Example:

```
SELECT pg_last_xlog_replay_location();
pg_last_xlog_replay_location
-----
0/2B16530
(1 row)
```

pg_last_xact_replay_timestamp()

Description: Gets the timestamp of last transaction replayed during restoration. This is the time to commit a transaction or abort a WAL record on the primary node. If no transactions have been replayed during restoration, this function will return **NULL**. Otherwise, if restoration is still in progress, this will increase monotonically. If restoration has completed, then this value will remain static at the value of the last WAL record received during that restoration. If the server normally starts without manual intervention, this function will return **NULL**.

Return type: timestamp with time zone

Restoration control functions control restoration processes. These functions may be executed only during restoration.

Example:

```
SELECT pg_last_xact_replay_timestamp();
pg_last_xact_replay_timestamp
-----
2023-01-04 07:03:08.098024+00
(1 row)
```

pg_is_xlog_replay_paused()

Description: Returns **true** if restoration is paused.

Return type: bool

pg_xlog_replay_pause()

Description: Pauses restoration immediately.

Return type: void

pg_xlog_replay_resume()

Description: Restarts restoration if it was paused.

While restoration is paused, no further database changes are applied. In hot standby mode, all new queries will see the same consistent snapshot of the database, and no further query conflicts will be generated until restoration is resumed.

If streaming replication is disabled, the paused state may continue indefinitely without problem. While streaming replication is in progress, WAL records will continue to be received, which will eventually fill available disk space. This progress depends on the duration of the pause, the rate of WAL generation, and available disk space.

Return type: void

pg_xlog_replay_completion()

Description: Displays the progress of xlog redo on the current DN.

Return type: record

Example:

```
SELECT * FROM pg_xlog_replay_completion();
replay_start | replay_current | replay_end | replay_percent
-----+-----+-----+-----
0/2ACAB80   | 0/2B16530     | 0/4F62B090 | 0%
(1 row)
```

The following information is returned:

Table 21-64 pg_xlog_replay_completion() columns

| Column | Type | Description |
|----------------|---------|--|
| replay_start | integer | Start LSN of xlog redo |
| replay_current | integer | LSN of the current replay of xlog redo |
| replay_end | integer | Maximum LSN that requires xlog redo |
| replay_percent | integer | Completion percentage of xlog redo |

pg_data_sync_from_dummy_completion()

Description: Displays the progress of data page file synchronization during the failover on the current DN.

Return type: record

Example:

```
SELECT * FROM pg_data_sync_from_dummy_completion();
start_index | current_index | total_index | sync_percent
-----+-----+-----+-----
0 | 0 | 0 | 100%
(1 row)
```

The following information is returned:

Table 21-65 pg_data_sync_from_dummy_completion() columns

| Column | Type | Description |
|---------------|---------|---|
| start_index | integer | Start LSN of data page file synchronization |
| current_index | integer | Current LSN of data page file synchronization |
| total_index | integer | Maximum LSN of data page file synchronization |
| sync_percent | integer | Completion percentage of data page files |

gs_roach_stop_backup(backupid text)

Description: Stops a backup started by the internal backup tool GaussRoach and returns the position where the current log is inserted. This function is similar to **pg_stop_backup**, but is more lightweight.

Return type: text

gs_roach_enable_delay_ddl_recycle(backupid name)

Description: Enables DDL delay and returns the log position of the enabling point. This function is similar to **pg_enable_delay_ddl_recycle**, but is more lightweight. In addition, this function allows you to enable DDL delay for multiple backups.

Return type: text

gs_roach_disable_delay_ddl_recycle(backupid text)

Description: Disables DDL delay, returns the logs for which DDL delay takes effect, and deletes the physical files of the column-store tables that have been deleted by the user. This function is similar to **pg_enable_delay_ddl_recycle**, but is more lightweight. In addition, this function allows you to disable DDL delay for multiple backups.

Return type: record

gs_roach_switch_xlog(request_ckpt bool)

Description: Switches the currently used log segment file and returns the position of the segment log. If the value of **request_ckpt** is **true**, a full check point is triggered.

Return type: text

pg_resume_bkp_flag(backupid name)

Description: This function resumes the **delay xlog** flag from a specified backup and returns **start_backup_flag boolean**, **to_delay boolean**, **ddl_delay_recycle_ptr text**, and **rewind_time text**.

Return type: record

pg_update_reloptions_in_newcluster

Description: This function writes the storage path in the file specified by the **obs-store-path-list** parameter to the **reloptions** field in the **pg_class** table during full restoration of V3 tables to a new cluster. This function is supported only by clusters of version 9.1.0 or later.

Return type: bool

21.7.29 Database Object Functions

21.7.29.1 Database Object Size Functions

Database object size functions calculate and return the actual disk space used by database objects in bytes.

pg_column_size(any)

Description: Specifies the number of bytes used to store a particular value (possibly compressed).

Return type: integer

Note: **pg_column_size** displays the space for storing an independent data value.

```
postgres=#SELECT pg_column_size(1);
pg_column_size
-----
         4
(1 row)
```

pg_database_size(oid)

Description: Specifies the disk space used by the database with the specified OID.

Return type: bigint

pg_database_size(name)

Description: Specifies the disk space used by the database with the specified name.

Return type: bigint

Note: **pg_database_size** receives the OID or name of a database and returns the disk space used by the corresponding object.

Example:

```
postgres=#SELECT pg_database_size('gaussdb');
pg_database_size
-----
      51590112
(1 row)
```

pg_relation_size(oid)

Description: Specifies the disk space used by the table with a specified OID or index.

Return type: bigint

get_db_source_datasize()

Description: Estimates the total size of non-compressed data in the current database.

Return type: bigint

Note: (1) **ANALYZE** must be performed before this function is called. (2) Calculate the total size of non-compressed data by estimating the compression rate of column-store tables.

Example:

```
postgres=#analyze;
ANALYZE
postgres=#SELECT get_db_source_datasize();
get_db_source_datasize
-----
      35384925667
(1 row)
```

pg_relation_size(text)

Description: Specifies the disk space used by the table with a specified name or index. The table name can be schema-qualified.

Return type: bigint

pg_relation_size(relation regclass, fork text)

Description: Specifies the disk space used by the specified bifurcating tree ('main', 'fsm', or 'vm') of a certain table or index.

Return type: bigint

pg_relation_size(relation regclass)

Description: Is an abbreviation of **pg_relation_size(..., 'main')**.

Return type: bigint

Note: **pg_relation_size** returns the size in bytes of the table, index, or compressed table with the specified OID or name.

pg_partition_size(oid,oid)

Description: Specifies the disk space used by the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.

Return type: bigint

pg_partition_size(text, text)

Description: Specifies the disk space used by the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.

Return type: bigint

pg_partition_indexes_size(oid,oid)

Description: Specifies the disk space used by the index of the partition with a specified OID. The first **oid** is the OID of the table and the second **oid** is the OID of the partition.

Return type: bigint

pg_partition_indexes_size(text,text)

Description: Specifies the disk space used by the index of the partition with a specified name. The first **text** is the table name and the second **text** is the partition name.

Return type: bigint

pg_indexes_size(regclass)

Description: Specifies the total disk space used by the index appended to the specified table.

Return type: bigint

pg_size_pretty(bigint)

Description: Converts the calculated byte size into a size readable to human beings.

Return type: text

pg_size_pretty(numeric)

Description: Converts the calculated byte size indicated by a numeral into a size readable to human beings.

Return type: text

Note: **pg_size_pretty** formats the results of other functions into a human-readable format. KB/MB/GB/TB can be used.

pg_table_size(regclass)

Description: Specifies the disk space used by the specified table, excluding indexes (but including TOAST, free space mapping, and visibility mapping).

Return type: bigint

pg_total_relation_size(oid)

Description: Specifies the disk space used by the table with a specified OID, including the index and the compressed data.

Return type: bigint

pg_total_relation_size(regclass)

Description: Specifies the total disk space used by the specified table, including all indexes and TOAST data.

Return type: bigint

pg_total_relation_size(text)

Description: This function specifies the disk space used by the table with the specified name, including the index and compressed data. The table name can be schema-qualified.

Return type: bigint

Note: **pg_total_relation_size** can specify the OID or name of a table or compressed table, and then return the data in bytes and the sizes of all related indexes and compressed tables.

pg_obs_file_size(regclass)

Description: This function obtains the CU file size, file name, and bucket ID stored on OBS by specifying the OID or name of a V3 column-store table. This function is supported only by clusters of version 9.1.0 or later.

Parameter: The input parameter can be the table OID or table name.

Return type: record

Table 21-66 Return fields

| Field | Type | Description |
|----------|---------|--|
| bucketid | integer | ID of the bucket for storing CU files. |
| filename | text | CU file name. |
| size | bigint | CU file size, in bytes. |

Example:

```
SELECT * FROM pg_obs_file_size('t3_col_part_dif_partition');
 bucketid | filename | size
-----+-----+-----
      9 | 23488119324673.0 | 512
     22 | 23488102760449.0 | 512
     21 | 23488119521281.0 | 512
     16 | 23488102662145.0 | 512
(4 rows)
```

pg_obs_file_size(text, text)

Description: This function specifies a partition of a column-store V3 table (primary table name or partitioned table) and returns the size, file name, and bucket ID of the CU file stored on OBS. This function is supported only by clusters of version 9.1.0 or later.

Parameter: The input parameter can be the table OID or table name. The partition name is used as the partition name.

Return type: record

| Field | Type | Description |
|----------|---------|--|
| bucketid | integer | ID of the bucket for storing CU files. |
| filename | text | CU file name. |
| size | bigint | CU file size, in bytes. |

21.7.29.2 Database Object Position Functions

pg_relation_filenode(relation regclass)

Description: Specifies the ID of a filenode with the specified relationship.

Return type: OID

Description: **pg_relation_filenode** receives the OID or name of a table, index, sequence, or compressed table, and returns the filenode number allocated to it. The filenode is the basic component of the file name used by the relationship. For most tables, the result is the same as that of **pg_class.relfilenode**. For the specified system directory, **relfilenode** is **0** and this function must be used to obtain the correct value. If a relationship that is not stored is transmitted, such as a view, this function returns **NULL**.

pg_relation_filepath(relation regclass)

Description: Specifies the name of a file path with the specified relationship.

Return type: text

Description: **pg_relation_filepath** is similar to **pg_relation_filenode**, except that **pg_relation_filepath** returns the whole file path name for the relationship (relative to the data directory **PGDATA** of the database cluster).

21.7.29.3 Partition Management Function

proc_add_partition (relname regclass, boundaries_interval interval)

Description: Adds partitions to a table with the automatic partition creation function enabled.

Return type: void

Note: When the function is executed, multiple partitions with time range as **boundaries_interval** are created based on the existing partition boundary until **new_part_boundary - now_time >= 29 * boundaries_interval**. Then, an extra partition is created to ensure that at least a new partition is created when the function is executed.

Example:

```
call proc_add_partition('my_schema.my_table', interval '1 day');
proc_add_partition
-----
(1 row)
```

proc_drop_partition (relname regclass, older_than interval)

Description: Deletes partitions from a table with the automatic partition deletion function enabled.

Return type: void

Note: When this function is executed, all partitions in the table are traversed and the partitions whose boundary is smaller than **now_time - older_than** are

deleted. If all partitions meet the deletion condition, the table is truncated with one partition kept.

Example:

```
call proc_drop_partition('my_schema.my_table', interval '7 day');
proc_drop_partition
-----
(1 row)
```

pg_partition_management_time(table_name, time)

Description: Modifies the execution time of the partition scheduling task that automatically adds or removes partitions. The default time is when the auto-increment partition table was created. This function is supported only by 8.3.0 and later versions.

Return value: **SUCCESS**, which indicates that the time is successfully modified.

Parameter:

- **table_name** indicates the name of the auto-increment partitioned table. The type is Name.
- **time** indicates the time when the data switching task is scheduled. The type is Time.

Example:

```
call pg_partition_management_time('partition_management_time','3:00');
pg_partition_management_time
-----
SUCCESS
(1 row)
```

21.7.29.4 Collation Version Function

pg_collation_actual_version (oid)

Description: Returns the actual version of the collation object currently installed in the operating system. Currently, this parameter is valid only for case_insensitive collations.

Return type: text

Example:

```
SELECT oid FROM pg_collation WHERE collname = 'case_insensitive';
oid
-----
3300
(1 row)

SELECT pg_collation_actual_version(3300);
pg_collation_actual_version
-----
153.14
(1 row)
```

21.7.29.5 Hot and Cold Table Functions

pg_obs_cold_refresh_time(table_name, time)

Description: Modifies the time when cold data in a hot and cold table is migrated to OBS. The default value is 00:00 every day.

table_name indicates the name of the hot and cold table, and the type is Name.
time indicates the time when the data switchover task is scheduled, and the type is Time.

Return value: **SUCCESS**, which indicates that the time is successfully modified.

Example:

```
SELECT * FROM pg_obs_cold_refresh_time('lifecycle_table', '06:30:00');
pg_obs_cold_refresh_time
-----
SUCCESS
(1 row)
```

pg_refresh_storage()

Description: Switches hot data to cold data on all hot and cold tables (in OBS).

Return type: int

Columns in the returned value:

- **success_count int**: indicates the number of tables that are successfully switched.
- **failed_count int**: indicates the number of tables that fail to be switched.

Example:

```
SELECT * FROM pg_refresh_storage();
success_count | failed_count
-----+-----
1 | 0
(1 row)
```

pg_lifecycle_table_data_distribute(table_name)

Description: Views the data distribution of a cold and hot table.

table_name indicates the table name and cannot be left blank.

Return value: record

Example: Multiple records are generated based on the number of nodes. The following example shows the data distribution in the **w1** table when there is only one DN node.

```
SELECT * FROM pg_catalog.pg_lifecycle_table_data_distribute('w1');
schemaname | tablename | nodename | hotpartition | coldpartition | switchablepartition | hotdatasize | colddatasize | switchabledatasize
-----+-----+-----+-----+-----+-----+-----+-----+-----
public | w1 | dn_1 | p2 | p1 | | 80 KB | 0 bytes | 0 bytes
(1 row)
```

pg_lifecycle_node_data_distribute()

Description: Views the data distribution of all hot and cold tables.

Return value: record

Example: There are two cold and hot tables in the database. The data distribution is as follows:

```
SELECT * FROM pg_catalog.pg_lifecycle_node_data_distribute();
 schemaname | tablename | nodename | hotpartition | coldpartition | switchablepartition | hotdatasize |
colddatasize | switchabledatasize
-----+-----+-----+-----+-----+-----+-----+-----
public      | w1        | dn_1     | p2           | p1            |                     | 81920      |
public      | w2        | dn_1     | p2           | p1            |                     | 81920      |
(2 rows)
```

refresh_hot_storage(relname text)

Description: Flushes all partition data of a specified cold and hot table to OBS. The returned value is the number of switched partitions. This function is supported only in 8.2.1.100 or later.

Parameter:

relname: table name (name of a cold and hot table. If the name of a non-cold and hot table is used, no error is reported and **0** is returned.)

Return type: integer

```
SELECT refresh_hot_storage('multi_temper_table');
 refresh_hot_storage
-----
4
(1 row)
```

refresh_hot_storage(relname text, partname text)

Description: Synchronizes partition data of a specified hot and cold table to OBS. The returned value is the number of switched partitions. This function is supported only in 8.2.1.100 or later.

Parameter:

- **relname:** table name (name of a cold and hot table. If the name of a non-cold and hot table is used, no error is reported and **0** is returned.)
- **partname:** partition name (specifies a partition name of a cold and hot table)

Return type: integer

```
SELECT refresh_hot_storage('multi_temper_table','p1');
 refresh_hot_storage
-----
1
(1 row)
```

reload_cold_partition(relname text)

Description: This function converts all cold partitions to hot ones. The returned value is the number of switched partitions. This function is supported only by 8.3.0 and later versions.

Parameter:

relname: table name (name of a cold and hot table. If the name of a non-cold and hot table is used, no error is reported and **0** is returned.)

Return type: integer

```
SELECT reload_cold_partition('multi_temper_table');
reload_cold_partition
-----
4
(1 row)
```

reload_cold_partition(relname text, partname text)

Description: This function converts specific cold partitions to hot ones. The returned value is the number of switched partitions. This function is supported only by 8.3.0 and later versions.

Parameter:

- **relname:** table name (name of a cold and hot table. If the name of a non-cold and hot table is used, no error is reported and **0** is returned.)
- **partname:** partition name (specifies a partition name of a cold and hot table)

Return type: integer

```
SELECT reload_cold_partition('multi_temper_table','p1');
reload_cold_partition
-----
1
(1 row)
```

21.7.30 Residual File Management Functions

21.7.30.1 Functions for Obtaining the Residual File List

pg_get_residualfiles()

Description: Obtains all residual file records of the current node. This function is an instance-level function and is irrelevant to the current database. It can run on any instance.

Return type: record

The following information is displayed:

Table 21-67 pg_get_residualfiles () return fields

| Column | Type | Description |
|--------------|------|-----------------|
| isverified | bool | Verified or not |
| isdeleted | bool | Deleted or not |
| dbname | text | Database name |
| residualfile | text | Data file path |

| Column | Type | Description |
|----------|------|--------------------|
| filepath | text | Residual file path |
| notes | text | Notes |

Example:

```
SELECT * FROM pg_get_residualfiles();
isverified | isdeleted | dbname | residualfile | filepath | notes
-----+-----+-----+-----+-----+-----
f          | f        | db2    | base/49155/114691 | pgrf_20200908160211441546 |
f          | f        | db2    | base/49155/114694 | pgrf_20200908160211441546 |
f          | f        | db2    | base/49155/114696 | pgrf_20200908160211441546 |
(3 rows)
```

pgxc_get_residualfiles()

Description: Unified CN query function of `pg_get_residualfiles()`. This function is a cluster-level function and is irrelevant to the current database. It runs on CNs.

Return type: record

The following information is displayed:

Table 21-68 `pgxc_get_residualfiles()` return fields

| Column | Type | Description |
|--------------|------|--------------------|
| nodename | text | Node name |
| isverified | bool | Verified or not |
| isdeleted | bool | Deleted or not |
| dbname | text | Database name |
| residualfile | text | Data file path |
| filepath | text | Residual file path |
| notes | text | Notes |

Example:

```
SELECT * FROM pgxc_get_residualfiles();
nodename | isverified | isdeleted | dbname | residualfile | filepath | notes
-----+-----+-----+-----+-----+-----+-----
cn_5001  | f          |          | postgres | base/15092/32803 | pgrf_20200910170129360401 |
dn_6001_6002 | f        | f        | db2    | base/49155/114691 | pgrf_20200908160211441546 |
dn_6001_6002 | f        | f        | db2    | base/49155/114694 | pgrf_20200908160211441546 |
dn_6001_6002 | f        | f        | db2    | base/49155/114696 | pgrf_20200908160211441546 |
(4 rows)
```

21.7.30.2 Functions for Scanning Residual Files

Only cluster 8.3.0 and later versions support the function of scanning residual files. All functions described in this section can be invoked only by the **sysadmin** role.

pg_scan_residualfiles()

Description: This function scans all residual file records in the database connected to the current node. It is an instance-level function and is related to the current database, and it can run on any instance.

- When it is executed on a CN, it scans the database of the CN and OBS for residual files.
- When it is executed on a CN, it scans the database of the DN for residual files.

Return type: record

The following information is displayed:

Table 21-69 Return fields

| Field | Type | Description |
|---------|------|---|
| pgscrfl | text | Local path of the metadata file that records residual file information. |

Example:

```
SELECT * FROM pg_scan_residualfiles();
pgscrfl
-----
pg_residualfiles/pgscrfl_meta_1663_16323_20231027143716005244
pg_residualfiles/pgscrfl_meta_2147484281_16323_20231027143716005713
(2 rows)
```

NOTE

The restrictions on using this function are as follows:

- Only the current database can be scanned for residual files. Currently, only the default tablespace, OBS tablespace, and user-defined tablespace are scanned.
- This function is a heavy-load operation and cannot be used concurrently on a single node. You should avoid using this function when the service is overloaded or the resource is scarce.
- Only hot partitions in cold-hot-partition-separated tables are supported. Cold partitions are not supported.
- Do not use this function during the upgrade observation period.
- Residual files in the deleted database cannot be scanned.
- Inconsistent cluster metadata information can cause this function incorrectly scan valid data files.

pgxc_scan_residualfiles(query_flag)

Description: Unified CN execution function of **pg_scan_residualfiles()** This function is a cluster-level function and is related to the current database. It runs on CNs.

Parameter description: **query_flag**. The parameter type is int, indicating the execution range. **1** indicates the CN, **2** indicates the primary DN, and **4** indicates the standby DN. The query union set can be obtained through the OR operation. For example, **1|2=3** indicates the CN and primary DN, and **1|2|4=7** indicates the CN, primary DN, and standby DN. The default value is **7**.

Return type: record

The command output is as follows.

Table 21-70 Return fields

| Field | Type | Description |
|-------------|------|---|
| nodename | name | Node name. |
| instance_id | text | Instance name. |
| pgscrfs | text | Local path of the metadata file that records residual file information. |

Example:

```
SELECT * FROM pgxc_scan_residualfiles();
 node_name | instance_id | pgscrfs
-----+-----+-----
dn_6001_6002 | dn_6001 | pg_residualfiles/pgscrfs_meta_1663_16323_20231027144103839354
dn_6001_6002 | dn_6001 | pg_residualfiles/pgscrfs_meta_2147484281_16323_20231027144103839826
cn_5001 | cn_5001 | pg_residualfiles/pgscrfs_meta_1663_16323_20231027144103946217
dn_6007_6008 | dn_6008 | pg_residualfiles/pgscrfs_meta_1663_16323_20231027144104171311
(4 rows)

SELECT * FROM pgxc_scan_residualfiles(1);
 node_name | instance_id | pgscrfs
-----+-----+-----
cn_5001 | cn_5001 | pg_residualfiles/pgscrfs_meta_1663_16323_20231027144421861628
(1 row)

SELECT * FROM pgxc_scan_residualfiles(2);
 node_name | instance_id | pgscrfs
-----+-----+-----
dn_6001_6002 | dn_6001 | pg_residualfiles/pgscrfs_meta_1663_16323_20231027144424210395
dn_6001_6002 | dn_6001 | pg_residualfiles/pgscrfs_meta_2147484281_16323_20231027144424210855
(2 rows)

SELECT * FROM pgxc_scan_residualfiles(4);
 node_name | instance_id | pgscrfs
-----+-----+-----
dn_6007_6008 | dn_6008 | pg_residualfiles/pgscrfs_meta_1663_16323_20231027144427492060
(1 row)
```

 NOTE

The restrictions on using this function are as follows:

- Only the current database can be scanned for residual files. Currently, only the default tablespace, OBS tablespace, and user-defined tablespace are scanned.
- This function is a heavy-load operation and cannot be used concurrently on a single node. You should avoid using this function when the service is overloaded or the resource is scarce.
- Only hot partitions in cold-hot-partition-separated tables are supported. Cold partitions are not supported.
- Do not use this function during the upgrade observation period.
- Residual files in the deleted database cannot be scanned.
- Inconsistent cluster metadata information can cause this function incorrectly scan valid data files.

pg_get_scan_residualfiles()

Description: Obtains all residual file records scanned on the current node. This function is an instance-level function and is irrelevant to the current database. It can run on any instance.

Return type: record

The command output is as follows.

Table 21-71 Return fields

| Field | Type | Description |
|--------------|------------|---|
| handled | bool | Whether residual files have been processed, moved, or modified. |
| dbname | text | Database name |
| residualfile | text | Path of the residual file |
| size | bigint | File size, in bytes. For residual files in the OBS path, the value of this parameter is 0 . |
| inode | bigint | Inode in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is 0 . |
| atime | timestampz | Access time in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is empty. |
| mtime | timestampz | Modify time in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is empty. |
| ctime | timestampz | Change time in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is empty. |

| Field | Type | Description |
|----------|------|--|
| filepath | text | Local path of the metadata file that records residual file information, corresponding to the pgscrfl_meta file. |
| notes | text | Notes |

pgxc_get_scan_residualfiles(query_flag)

Description: Unified CN execution function of **pg_get_scan_residualfiles()** This function is a cluster-level function and is irrelevant to the current database. It runs on CNs.

The **query_flag** parameter is of the int type and indicates the execution range. **1** indicates the CN, **2** indicates the primary DN, and **4** indicates the standby DN. The query union set can be obtained through the OR operation. For example, **1|2=3** indicates the CN and primary DN, and **1|2|4=7** indicates the CN, primary DN, and standby DN. The default value is **7**.

Return type: record

The command output is as follows.

Table 21-72 Return fields

| Field | Type | Description |
|--------------|------------|---|
| nodename | name | Node name |
| instance_id | text | Instance name |
| handled | bool | Whether residual files have been processed, moved, or modified. |
| dbname | text | Database name |
| residualfile | text | Path of the residual file |
| size | bigint | File size, in bytes. For residual files in the OBS path, the value of this parameter is 0 . |
| inode | bigint | Inode in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is 0 . |
| atime | timestampz | Access time in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is empty. |
| mtime | timestampz | Modify time in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is empty. |

| Field | Type | Description |
|----------|------------|--|
| ctime | timestampz | Change time in the stat information of the residual file. For residual files in the OBS path, the value of this parameter is empty. |
| filepath | text | Local path of the metadata file that records residual file information, corresponding to the pgscrfl_meta file. |
| notes | text | Notes |

pg_archive_scan_residualfiles()

Description: Archives all residual file records scanned on the database connected to the current node. This function is an instance-level function and is related to the current database. It can run on any instance.

Return type: record

The command output is as follows.

Table 21-73 Return fields

| Field | Type | Description |
|---------|--------|--|
| archive | text | Path of the local folder after archiving. Residual files in the OBS path are archived in the corresponding OBS database directory. |
| count | bigint | Number of archived residual files. |
| size | bigint | Total size of archived residual files, in bytes. |

Example:

```
SELECT * FROM pg_archive_scan_residualfiles();
      archive      | count | size
-----+-----+-----
pg_residualfiles/archive/pgscrfl_archive_20231027144613791801 |    4 | 81920
(1 row)
```

 NOTE

- This function can archive only the residual files recorded in the current login database. During archiving, the residual files are verified. The verification results are as follows:
 - Verification passed: After the verification is passed, the residual files are archived and marked as handled.
 - Verification failed: If the verification fails, the residual files are not archived and marked as handled.
 - Verification delayed: If the verification is delayed, it indicates that verification is not possible at the moment, and archiving is skipped. The verification delay is related to the transaction completion status and the standby node's redo progress.
- The actual archive directory and the corresponding tablespace are in the same file system. Deleting the tablespace also deletes the archived residual files.
- This function cannot be used after DDL delay is enabled.
- This function is a heavy-load operation and cannot be used concurrently on a single node. You should avoid using this function when the service is overloaded or the resource is scarce.

pgxc_archive_scan_residualfiles(query_flag)

Description: Unified CN execution function of **pg_archive_scan_residualfiles()**

This function is a cluster-level function and is related to the current database. It runs on CNs.

The **query_flag** parameter is of the int type and indicates the execution range. **1** indicates the CN, **2** indicates the primary DN, and **4** indicates the standby DN. The query union set can be obtained through the OR operation. For example, **1|2=3** indicates the CN and primary DN, and **1|2|4=7** indicates the CN, primary DN, and standby DN. The default value is **7**.

Return type: record

The command output is as follows.

Table 21-74 Return fields

| Field | Type | Description |
|-------------|--------|--|
| nodename | name | Node name. |
| instance_id | text | Instance name. |
| archive | text | Path of the local folder after archiving. Residual files in the OBS path are archived in the corresponding OBS database directory. |
| count | bigint | Number of archived residual files. |
| size | bigint | Total size of archived residual files, in bytes. |

Example:

```
SELECT * FROM pgxc_archive_scan_residualfiles();
```

```
node_name | instance_id | archive | count | size
```

```
cn_5001 | cn_5001 | pg_residualfiles/archive/pgscrif_archive_20231027145050896440 | 1 | 40960  
dn_6007_6008 | dn_6008 | pg_residualfiles/archive/pgscrif_archive_20231027145051018138 | 1 | 24576  
(2 rows)
```

NOTE

- This function can archive only the residual files recorded in the current login database. During archiving, the residual files are verified. The verification results are as follows:
 - Verification passed: After the verification is passed, the residual files are archived and marked as handled.
 - Verification failed: If the verification fails, the residual files are not archived and marked as handled.
 - Verification delayed: If the verification is delayed, it indicates that verification is not possible at the moment, and archiving is skipped. The verification delay is related to the transaction completion status and the standby node's redo progress.
- The actual archive directory and the corresponding tablespace are in the same file system. Deleting the tablespace also deletes the archived residual files.
- This function cannot be used after DDL delay is enabled.
- This function is a heavy-load operation and cannot be used concurrently on a single node. You should avoid using this function when the service is overloaded or the resource is scarce.

pg_rm_scan_residualfiles_archive()

Description: Deletes all archived residual file records on the current node. This function is an instance-level function and is irrelevant to the current database. It can run on any instance.

Return type: record

The command output is as follows.

Table 21-75 Return fields

| Field | Type | Description |
|-------|--------|---|
| count | bigint | Number of residual files that have been deleted from the archive. For residual files in the local path, the number of deleted files is counted. For residual files in the OBS path, the number of deleted table directories is counted. |
| size | bigint | Total size of residual files that have been deleted from the archive, in bytes. For residual files in the OBS path, the value of this parameter is 0 . |

Example:

```
SELECT * FROM pg_rm_scan_residualfiles_archive();  
count | size  
-----+-----  
4 | 81920  
(1 row)
```

pgxc_rm_scan_residualfiles_archive(query_flag)

Description: Unified CN execution function of **pg_rm_scan_residualfiles_archive()**. This function is a cluster-level function and is irrelevant to the current database. It runs on CNs.

The **query_flag** parameter is of the int type and indicates the execution range. **1** indicates the CN, **2** indicates the primary DN, and **4** indicates the standby DN. The query union set can be obtained through the OR operation. For example, **1|2=3** indicates the CN and primary DN, and **1|2|4=7** indicates the CN, primary DN, and standby DN. The default value is **7**.

Return type: record

The command output is as follows.

Table 21-76 Return fields

| Field | Type | Description |
|-------------|--------|---|
| nodename | name | Node name. |
| instance_id | text | Instance name. |
| count | bigint | Number of residual files that have been deleted from the archive. For residual files in the local path, the number of deleted files is counted. For residual files in the OBS path, the number of deleted table directories is counted. |
| size | bigint | Total size of residual files that have been deleted from the archive, in bytes. For residual files in the OBS path, the value of this parameter is 0 . |

Example:

```
SELECT * FROM pgxc_rm_scan_residualfiles_archive();
 node_name | instance_id | count | size
-----+-----+-----+-----
dn_6001_6002 | dn_6001 | 4 | 81920
cn_5001 | cn_5001 | 1 | 40960
dn_6007_6008 | dn_6008 | 1 | 24576
coordinator1 | coordinator1 | 1 | 0
(4 rows)
```

21.7.30.3 Functions for Verifying Residual Files

pg_verify_residualfiles(filepath)

Description: Verifies whether the file recorded in the parameter specified file is a residual file. This function is an instance-level function and is related to the current database. It can run on any instance.

Parameter type: text

Return type: bool

The following table describes return columns.

Table 21-77 pg_verify_residualfiles (filepath) return fields

| Column | Type | Description |
|------------|------|---------------------------------------|
| isverified | bool | Whether the verification is complete. |

Example:

```
postgres=#SELECT * FROM pg_verify_residualfiles('pgrf_20200908160211441546');
isverified
-----
t
(1 row)
```

 NOTE

This function only verifies whether the recorded file is a residual file in the current database. If the recorded file is not in the current database, the verification is not applicable.

pg_verify_residualfiles()

Description: Verifies whether recorded files on all residual file lists of the current instance are residual files. This function is an instance-level function and is related to the current database. It can run on any instance.

Parameter type: none

Return type: record

The following table describes return columns.

Table 21-78 pg_verify_residualfiles () return fields

| Column | Type | Description |
|----------|------|---------------------------------------|
| result | bool | Whether the verification is complete. |
| filepath | text | Residual file path. |
| notes | text | Notes. |

Example:

```
postgres=#SELECT * FROM pg_verify_residualfiles();
result |      filepath      | notes
-----+-----+-----
t      | pgrf_20200908160211441546 |
(1 row)
```

 NOTE

This function only verifies whether the recorded file is a residual file in the current database. If the recorded file is not in the current database, the verification is not applicable.

pgxc_verify_residualfiles()

Description: Unified CN query function of pg_verify_residualfiles() This function is a cluster-level function and is related to the current database. It runs on CNs.

Parameter type: none

Return type: record

The following table describes return columns.

Table 21-79 pgxc_verify_residualfiles () return fields

| Column | Type | Description |
|----------|------|---------------------------------------|
| nodename | text | Node name. |
| result | bool | Whether the verification is complete. |
| filepath | text | Residual file path. |
| notes | text | Notes. |

Example:

```
postgres=#SELECT * FROM pgxc_verify_residualfiles();
 nodename | result |      filepath      | notes
-----+-----+-----+-----
  cn_5001 |    t   | pgrf_20200910170129360401 |
dn_6001_6002 | t     | pgrf_20200908160211441546 |
(2 rows)
```

NOTE

This function only verifies whether the recorded file is a residual file in the current database. If the recorded file is not in the current database, the verification is not applicable.

pg_is_residualfiles(residualfile)

Description: Queries whether a specified **relfilenode** is a residual file in the current database. This function is an instance-level function and is related to the current database. It can run on any instance.

Parameter type: text

Return type: bool

The following table describes return columns.

Table 21-80 pg_is_residualfiles (residualfile) return fields

| Column | Type | Description |
|--------|------|------------------------------------|
| result | bool | Whether a file is a residual file. |

Example:

```
postgres=#SELECT * FROM pg_is_residualfiles('base/49155/114691');
result
-----
t
(1 row)
```

NOTE

This function only verifies whether the recorded file is a residual file in the current database. If the recorded file is not in the current database, it is verified as a residual file.

For example, the file **base/15092/14790** is not regarded as a residual file in a **gaussdb** database, but it is regarded as a residual file in other databases.

```
postgres=#SELECT * FROM pg_is_residualfiles('base/15092/14790');
result
-----
f
(1 row)

postgres=#\c db2
db2=# SELECT * FROM pg_is_residualfiles('base/15092/14790');
result
-----
t
(1 row)
```

21.7.30.4 Functions for Deleting Residual Files

pg_rm_residualfiles(filepath)

Description: Deletes files from a specified residual file list on the current instance. This function is an instance-level function and is irrelevant to the current database. It can run on any instance.

Parameter type: text

Return type: record

The following table describes return columns.

Table 21-81 pg_rm_residualfiles (filepath) return fields

| Column | Type | Description |
|--------|------|---------------------------|
| result | bool | Deletion completed or not |

Example:

```
postgres=#SELECT * FROM pg_rm_residualfiles('pgrf_20200908160211441599');
result
-----
t
(1 row)
```


 NOTE

- Residual files can be deleted only after verification using the `pg_verify_residualfiles()` function.
- All verified files, regardless which database they are in, will be deleted.
- If all files recorded in the specified file have been deleted, the specified file will be removed and backed up in the `$PGDATA/pg_residualfile/backup` directory.

`pg_rm_residualfiles()`

Description: Deletes all files recorded on all residual file lists on the current instance. This function is an instance-level function and is irrelevant to the current database. It can run on any instance.

Parameter type: none

Return type: record

The following table describes return columns.

Table 21-82 `pg_rm_residualfiles ()` return fields

| Column | Type | Description |
|----------|------|---------------------------|
| result | bool | Deletion completed or not |
| filepath | text | Residual file path |
| notes | text | Notes |

Example:

```
postgres=#SELECT * FROM pg_rm_residualfiles();
result |      filepath      | notes
-----+-----+-----
t      | pg_rf_20200908160211441546 |
(1 row)
```

 NOTE

- Residual files can be deleted only after verification using the `pg_verify_residualfiles()` function.
- All verified files, regardless which database they are in, will be deleted.
- If all files recorded in the specified file have been deleted, the specified file will be removed and backed up in the `$PGDATA/pg_residualfile/backup` directory.

`pgxc_rm_residualfiles()`

Description: Unified CN query function of `pgxc_rm_residualfiles`. This function is a cluster-level function and is irrelevant to the current database. It runs on CNs.

Parameter type: none

Return type: record

The following table describes return columns.

Table 21-83 pgxc_rm_residualfiles () return fields

| Column | Type | Description |
|----------|------|---------------------------|
| nodename | text | Node name |
| result | bool | Deletion completed or not |
| filepath | text | Residual file path |
| notes | text | Notes |

Example:

```
postgres=#SELECT * FROM pgxc_rm_residualfiles();
 nodename | result |      filepath      | notes
-----+-----+-----+-----
  cn_5001 |    t   | pgrf_20200910170129360401 |
 dn_6001_6002 | t     | pgrf_20200908160211441546 |
(2 rows)
```

pgxc_clear_disk_cache()

Description: This function deletes all disk cache files. This function is supported only by clusters of version 9.0.2 or later.

Return type: void

Example:

```
SELECT pgxc_clear_disk_cache();
pgxc_clear_disk_cache
-----
(1 row)
```

21.7.30.5 Using Residual File Management Functions

Procedure

- Step 1** Call the **pgxc_get_residualfiles()** function to obtain the name of the database that has residual files.
- Step 2** Go to the databases where residual files exist and call the **pgxc_verify_residualfiles()** function to verify the residual files recorded in the current database.
- Step 3** Call the **pgxc_rm_residualfiles()** function to delete all the verified residual files.

----End

NOTE

The pgxc residual file management function only operates on the CN and the current primary DN, and does not verify or clear residual files on the standby DN. Therefore, after the primary DN is cleared, you need to clear residual files on the standby DN or build the standby DN in a timely manner. This prevents residual files on the standby DN from being copied back to the primary DN due to incremental build after a primary/standby switchover.

Examples

The following example uses two user-created databases, **db1** and **db2**.

```
postgres=# \l
```

| Name | Owner | Encoding | Collate | Ctype | Access privileges |
|-----------|--------|-----------|---------|-------|-------------------|
| db1 | fpi810 | SQL_ASCII | C | C | |
| db2 | fpi810 | SQL_ASCII | C | C | |
| postgres | fpi810 | SQL_ASCII | C | C | |
| template0 | fpi810 | SQL_ASCII | C | C | =c/fpi810 + |
| | | | | | fpi810=CTc/fpi810 |
| template1 | fpi810 | SQL_ASCII | C | C | =c/fpi810 + |
| | | | | | fpi810=CTc/fpi810 |

(5 rows)

- Run the following command to obtain all residual file records of the cluster on the CNs:

```
db1=# select * from pgxc_get_residualfiles() order by 4, 6; -- order by is optional.
```

```
db1=# select * from pgxc_get_residualfiles() order by 4, 6;
```

| nodename | isverified | isdeleted | dbname | residualfile | filepath | notes |
|--------------|------------|-----------|--------|------------------|---------------------------|-------|
| dn_6001_6002 | f | f | db1 | base/16390/16395 | pgrf_20200921153355979438 | |
| dn_6001_6002 | f | f | db1 | base/16390/24592 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24579 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24582 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24584 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24585 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24588 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24589 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24591 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24576 | pgrf_20200921153612088342 | |

(10 rows)

In the current cluster:

- Residual file records exist in the **db1** and **db2** databases on the **dn_6001_6002** node (active node instance).
 - Residual files are displayed in the **residualfile** column.
 - The **filepath** column lists the files that record residual files. These files are stored in the **pg_residualfiles** directory under the instance data directory.
- Call the **pgxc_verify_residualfiles()** function to verify the **db1** database.

```
db1=# SELECT * FROM pgxc_verify_residualfiles();
```

```
postgres=# \c db1
```

Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "db1" as user "fpi810".

```
db1=# select * from pgxc_verify_residualfiles();
```

| nodename | result | filepath | notes |
|--------------|--------|---------------------------|-------|
| dn_6001_6002 | t | pgrf_20200921153355979438 | |
| dn_6001_6002 | t | pgrf_20200921153612088342 | |

(2 rows)

Verification functions are at the database level. Therefore, when a verification function is called in the **db1** database, it only verifies residual files in **db1**.

You can call the **get** function again to check whether the verification is complete.

```
db1=# SELECT * FROM pgxc_get_residualfiles() order by 4, 6;
```

```
db1=# select * from pgxc_get_residualfiles() order by 4, 6;
```

| nodename | isverified | isdeleted | dbname | residualfile | filepath | notes |
|--------------|------------|-----------|--------|------------------|---------------------------|-------|
| dn_6001_6002 | t | f | db1 | base/16390/16395 | pgrf_20200921153355979438 | |
| dn_6001_6002 | t | f | db1 | base/16390/24592 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24579 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24582 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24584 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24585 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24588 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24589 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24591 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24576 | pgrf_20200921153612088342 | |

(10 rows)

As shown in the preceding figure, the residual files in the **db1** database have been verified, and the residual files in the **db2** database are not verified.

3. Call the **pgxc_rm_residualfiles()** function to delete residual files.

```
db1=# SELECT * FROM pgxc_rm_residualfiles();
```

```
db1=# select * from pgxc_rm_residualfiles();
```

| nodename | result | filepath | notes |
|--------------|--------|---------------------------|-------|
| dn_6001_6002 | t | pgrf_20200921153355979438 | |
| dn_6001_6002 | t | pgrf_20200921153612088342 | |

(2 rows)

4. Call the **pgxc_get_residualfiles()** function again to check the deletion result.

```
db1=# select * from pgxc_get_residualfiles() order by 4, 6;
```

| nodename | isverified | isdeleted | dbname | residualfile | filepath | notes |
|--------------|------------|-----------|--------|------------------|---------------------------|-------|
| dn_6001_6002 | t | t | db1 | base/16390/24592 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24579 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24582 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24584 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24576 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24588 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24589 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24591 | pgrf_20200921153612088342 | |
| dn_6001_6002 | f | f | db2 | base/16391/24585 | pgrf_20200921153612088342 | |

(9 rows)

The result shows that the residual files in the **db1** database are deleted (**isdeleted** is marked as **t**) and the residual files in the **db2** database are not deleted.

In addition, nine query results are displayed. Compared with the previous query results, a record for the residual file ending with **9438** is missing. This is because the record file that records the residual file ending with **9438** contains only one record, which is deleted in step 3. If all residual files in a record file are deleted, the record file is also deleted. Deleted files are backed up in the **pg_residualfiles/backup** directory.

```
[fpi810@host-192-168-244-162 pg_residualfiles]$ ls
backup pendingdeletesfile pgrf_20200921153612088342
[fpi810@host-192-168-244-162 pg_residualfiles]$ ls backup/
pgrf_20200921153355979438_bak
```

5. To delete files from the **db2** database, you need to call the **verify** function in the **db2** database and then call the **rm** function.
 - a. Go to the **db2** database and call the verification function.

```

db1=# \c db2
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "db2" as user "fpi810".
db2=# select * from pgxc_verify_residualfiles();
  nodename | result | filepath | notes
-----+-----+-----+-----
 dn_6001_6002 | t      | pgrf_20200921153612088342 |
(1 row)

```

Query the verification result:

```

db2=# select * from pgxc_get_residualfiles() order by 4, 6;
  nodename | isverified | isdeleted | dbname | residualfile | filepath | notes
-----+-----+-----+-----+-----+-----+-----
 dn_6001_6002 | t          | t         | db1     | base/16390/24592 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24579 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24582 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24584 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24576 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24588 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24589 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24591 | pgrf_20200921153612088342 |
 dn_6001_6002 | t          | f         | db2     | base/16391/24585 | pgrf_20200921153612088342 |
(9 rows)

```

- b. Call the deletion function:

```

db2=# select * from pgxc_rm_residualfiles();
  nodename | result | filepath | notes
-----+-----+-----+-----
 dn_6001_6002 | t      | pgrf_20200921153612088342 |
(1 row)

```

- c. Query the deletion result:

```

db2=# select * from pgxc_get_residualfiles() order by 4, 6;
  nodename | isverified | isdeleted | dbname | residualfile | filepath | notes
-----+-----+-----+-----+-----+-----+-----
(0 rows)

```

All residual files recorded in the record file whose name ends with **8342** have been deleted, so the record file is deleted and backed up in the **backup** directory. As a result, no records are found.

```

[fpi810@host-192-168-244-162 pg_residualfiles]$ ls
backup pendingdeletesfile
[fpi810@host-192-168-244-162 pg_residualfiles]$ ls backup/
pgrf_20200921153355979438_bak pgrf_20200921153612088342_bak

```

21.7.31 Statistics Information Functions

Statistics information functions are divided into the following two categories: functions that access databases, using the OID of each table or index in a database to mark the database for which statistics are generated; functions that access servers, identified by the server process ID, whose value ranges from 1 to the number of currently active servers.

For a global temporary table, the input parameter of the function is the OID of the table in **pg_class**. The function returns the statistics of the current session. The results of querying the same global temporary table in different sessions may be different.

pg_stat_get_db_numbackends(oid)

Description: Obtains the number of active server threads of a specified database on the current instance.

Return type: integer

pg_stat_get_db_total_numbackends(oid)

Description: Obtains the total number of active server threads of a specified database on all CNs in a cluster (if this function is executed on a CN), or obtains the number of active server threads of a specified database on the current instance (if this function is executed on a DN).

Return type: integer

pg_stat_get_db_xact_commit(oid)

Description: Obtains the number of committed transactions in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_xact_commit(oid)

Description: Obtains the total number of committed transactions in a specified database on all CNs in a cluster (if this function is executed on a CN), or obtains the number of committed transactions in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_xact_rollback(oid)

Description: Obtains the number of rollback transactions in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_xact_rollback(oid)

Description: Obtains the total number of rollback transactions in a specified database on all CNs in a cluster (if this function is executed on a CN), or obtains the number of rollback transactions in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_blocks_fetched(oid)

Description: Obtains the number of disk block fetch requests in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_blocks_fetched(oid)

Description: Obtains the total number of disk block fetch requests in a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of disk block fetch requests in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_blocks_hit(oid)

Description: Obtains the number of requested disk blocks found in the cache in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_blocks_hit(oid)

Description: Obtains the total number of requested disk blocks found in the cache in a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of requested disk blocks found in the cache in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_tuples_returned(oid)

Description: This function obtains the number of tuples returned for a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_tuples_returned(oid)

Description: Obtains the total number of tuples returned for a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of tuples returned for a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_tuples_fetched(oid)

Description: Obtains the number of tuples read from a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_tuples_fetched(oid)

Description: Obtains the total number of tuples read from a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of tuples read from a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_tuples_inserted(oid)

Description: Obtains the number of tuples inserted into a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_tuples_inserted(oid)

Description: Obtains the total number of tuples inserted into a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of tuples inserted into a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_tuples_updated(oid)

Description: Obtains the number of updated tuples in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_tuples_updated(oid)

Description: Obtains the total number of updated tuples in a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of updated tuples in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_tuples_deleted(oid)

Description: Obtains the number of tuples deleted from a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_tuples_deleted(oid)

Description: Obtains the total number of tuples deleted from a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of tuples deleted from a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_conflict_lock(oid)

Description: Obtains the total number of conflicting locks in a specified database on all CNs and DNs in a cluster (if this function is executed on a CN), or obtains the number of conflicting locks in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_deadlocks(oid)

Description: Obtains the number of deadlocks in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_deadlocks(oid)

Description: Obtains the total number of deadlocks in a specified database on all CNs and DN in a cluster (if this function is executed on a CN), or obtains the number of deadlocks in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_conflict_all(oid)

Description: Obtains the number of conflict recoveries in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_conflict_all(oid)

Description: Obtains the total number of conflict recoveries in a specified database on all CNs and DN in a cluster (if this function is executed on a CN), or obtains the number of conflict recoveries in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_temp_files(oid)

Description: Obtains the number of temporary files created in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_temp_files(oid)

Description: Obtains the total number of temporary files created in a specified database on all DN in a cluster (if this function is executed on a CN), or obtains the number of temporary files created in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_temp_bytes(oid)

Description: Obtains the number of bytes of the temporary files created in a specified database on the current instance.

Return type: bigint

pg_stat_get_db_total_temp_bytes(oid)

Description: Obtains the total number of bytes of the temporary files created in a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the number of bytes of the temporary files created in a specified database on the current instance (if this function is executed on a DN).

Return type: bigint

pg_stat_get_db_blk_read_time(oid)

Description: Obtains the time required for reading data blocks from a specified database on the current instance.

Return type: double

pg_stat_get_db_total_blk_read_time(oid)

Description: Obtains the total time required for reading data blocks from a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the time required for reading data blocks from a specified database on the current instance (if this function is executed on a DN).

Return type: double

pg_stat_get_db_blk_write_time(oid)

Description: Obtains the time required for writing data blocks to a specified database on the current instance.

Return type: double

pg_stat_get_db_total_blk_write_time(oid)

Description: Obtains the total time required for writing data blocks to a specified database on all DNs in a cluster (if this function is executed on a CN), or obtains the time required for writing data blocks to a specified database on the current instance (if this function is executed on a DN).

Return type: double

pg_stat_get_numscans(oid)

Description: Number of sequential row scans done if parameters are in a table or number of index scans done if parameters are in an index

Return type: bigint

pg_stat_get_tuple()

Description: This function can be executed on both CNs and DNs. This function is supported only by version 8.1.3 or later clusters.

If no parameters are specified, this function queries the statistics of all system catalogs on CNs, the dirty page rate of the tables on each CN, the statistics of all

system catalogs and user catalogs on DNs, and the dirty page rate of the tables on each DN.

If the schema name and table name are specified, this function queries the statistics and dirty page rate of the specified table.

 NOTE

The statistics of this function depend on the **ANALYZE** operation. To obtain the most accurate information, perform the **ANALYZE** operation on the table first.

Return type: record

The following table describes return columns.

| Column | Type | Description |
|-------------------|--------------------------|--|
| nodename | text | Node name |
| tableid | oid | Table OID |
| partid | oid | Partition OID of the partitioned table |
| last_vacuum | timestamp with time zone | Time of the last manual VACUUM |
| last_autovacuum | timestamp with time zone | Time of the last AUTOVACUUM |
| last_analyze | timestamp with time zone | Time of the last manual ANALYZE |
| last_autoanalyze | timestamp with time zone | Time of the last AUTOANALYZE |
| vacuum_count | bigint | Number of VACUUM operations |
| autovacuum_count | bigint | Number of AUTOVACUUM operations |
| analyze_count | bigint | Number of ANALYZE operations |
| autoanalyze_count | bigint | Number of times of automatically performing the ANALYZE operation |
| n_tup_ins | bigint | Number of rows inserted |
| n_tup_upd | bigint | Number of rows updated |
| n_tup_del | bigint | Number of rows deleted |
| n_tup_hot_upd | bigint | Number of rows with HOT updates |
| n_tup_change | bigint | Number of changed rows after ANALYZE |
| n_live_tup | bigint | Estimated number of live rows |
| n_dead_tup | bigint | Estimated number of dead rows |
| dirty_rate | bigint | Dirty page rate of a single CN or DN |

| Column | Type | Description |
|-------------------|--------------------------|--|
| last_data_changed | timestamp with time zone | Time when a table was last modified. This column is left empty in 8.2.0 and later versions. You are advised to query the value using pg_stat_get_last_data_changed_time(oid) . |

pg_stat_get_tuples_returned(oid)

Description: Number of sequential row scans done if parameters are in a table or number of index entries returned if parameters are in an index

Return type: bigint

pg_stat_get_tuples_fetched(oid)

Description: Number of table rows fetched by bitmap scans if parameters are in a table,

or table rows fetched by simple index scan using the index if parameters are in an index

Return type: bigint

pg_stat_get_tuples_inserted(oid)

Description: Number of rows inserted into table

Return type: bigint

pg_stat_get_local_tuples_inserted(oid)

Description: Number of rows inserted into the table on the current node. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_tuples_updated(oid)

Description: Number of rows updated in table

Return type: bigint

pg_stat_get_local_tuples_updated(oid)

Description: Number of rows updated in the table on the current node. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_tuples_deleted(oid)

Description: Number of rows deleted from table

Return type: bigint

pg_stat_get_local_tuples_deleted(oid)

Description: Number of rows deleted from the table on the current node. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_tuples_changed(oid)

Description: Queries the function on a CN and returns the total number of inserted, updated, and deleted rows in the table since the last **ANALYZE** or **AUTOANALYZE** operation. Queries the function on a DN and returns the total number of inserted, updated, and deleted rows in the table since the last **ANALYZE** or **AUTOANALYZE** operation on the current node.

Return type: bigint

pg_stat_get_local_tuples_changed(oid)

Description: Number of inserted, updated, and deleted rows in the table since the last **ANALYZE** or **AUTOANALYZE** operation on the current node.

Return type: bigint

pg_stat_get_tuples_hot_updated(oid)

Description: Number of rows HOT-updated in table

Return type: bigint

pg_stat_get_local_tuples_hot_updated(oid)

Description: Number of rows with HOT updates in the table on the current node. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_live_tuples(oid)

Description: Number of live tuples in the table.

Return type: bigint

pg_stat_get_local_live_tuples(oid)

Description: Number of live tuples in the table on the current node. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_dead_tuples(oid)

Description: Number of dead tuples in the table.

Return type: bigint

pg_stat_get_local_dead_tuples(oid)

Description: Number of dead tuples in the table on the current node. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_blocks_fetched(oid)

Description: Number of disk block fetch requests for table or index

Return type: bigint

pg_stat_get_blocks_hit(oid)

Description: Number of disk block requests found in cache for table or index

Return type: bigint

pg_stat_get_partition_tuples_inserted(oid)

Description: Number of rows in the corresponding table partition

Return type: bigint

pg_stat_get_partition_tuples_updated(oid)

Description: Number of rows that have been updated in the corresponding table partition

Return type: bigint

pg_stat_get_partition_tuples_deleted(oid)

Description: Number of rows deleted from the corresponding table partition

Return type: bigint

pg_stat_get_partition_tuples_changed(oid)

Description: Total number of inserted, updated, and deleted rows after the table partition was last analyzed or autoanalyzed

Return type: bigint

pg_stat_get_partition_live_tuples(oid)

Description: Number of live rows in a table partition

Return type: bigint

pg_stat_get_partition_dead_tuples(oid)

Description: Number of dead rows in a table partition

Return type: bigint

pg_stat_get_xact_tuples_inserted(oid)

Description: Number of tuple inserted into the active subtransactions related to the table.

Return type: bigint

pg_stat_get_xact_tuples_deleted(oid)

Description: Number of deleted tuples in the active subtransactions related to a table

Return type: bigint

pg_stat_get_xact_tuples_hot_updated(oid)

Description: Number of hot updated tuples in the active subtransactions related to a table

Return type: bigint

pg_stat_get_xact_tuples_updated(oid)

Description: Number of updated tuples in the active subtransactions related to a table

Return type: bigint

pg_stat_get_xact_partition_tuples_inserted(oid)

Description: Number of inserted tuples in the active subtransactions related to a table partition

Return type: bigint

pg_stat_get_xact_partition_tuples_deleted(oid)

Description: Number of deleted tuples in the active subtransactions related to a table partition

Return type: bigint

pg_stat_get_xact_partition_tuples_hot_updated(oid)

Description: Number of hot updated tuples in the active subtransactions related to a table partition

Return type: bigint

pg_stat_get_xact_partition_tuples_updated(oid)

Description: Number of updated tuples in the active subtransactions related to a table partition

Return type: bigint

pg_stat_get_last_vacuum_time(oid)

Description: Last time when the autovacuum thread is manually started to clear a table

Return type: timestamptz

pg_stat_get_last_autovacuum_time(oid)

Description: Time of the last vacuum initiated by the autovacuum thread on this table

Return type: timestamptz

pg_stat_get_local_last_autovacuum_time(oid)

Description: Time of the last vacuum initiated by the autovacuum thread of the current node on this table. This function is supported only in 8.1.2 or later.

Return type: timestamptz

pg_stat_get_vacuum_count(oid)

Description: Number of times a table is manually cleared

Return type: bigint

pg_stat_get_autovacuum_count(oid)

Description: Number of times of vacuum initiated by the autovacuum thread on this table

Return type: bigint

pg_stat_get_local_autovacuum_count(oid)

Description: Number of times of vacuum initiated by the autovacuum thread of the current node on this table. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_last_analyze_time(oid)

Description: Last time when a table starts to be analyzed manually or by the autovacuum thread

Return type: timestamptz

pg_stat_get_last_autoanalyze_time(oid)

Description: Time of the last analysis initiated by the autovacuum thread on this table

Return type: timestamptz

pg_stat_get_local_last_autoanalyze_time(oid)

Description: Time of the last analysis initiated by the autovacuum thread of the current node on this table. This function is supported only in 8.1.2 or later.

Return type: timestamptz

pg_stat_get_analyze_count(oid)

Description: Number of times a table is manually analyzed

Return type: bigint

pg_stat_get_autoanalyze_count(oid)

Description: Number of times the autovacuum daemon analyzes a table

Return type: bigint

pg_stat_get_local_autoanalyze_count(oid)

Description: Number of times that the autovacuum daemon of the current node starts analysis on this table. This function is supported only in 8.1.2 or later.

Return type: bigint

pg_stat_get_local_analyze_status(oid)

Description: Specifies whether to analyze the status of a table on the current node. This parameter is valid only for CNs. This function is supported only in 8.1.2 or later.

- If the number of modified rows in the table exceeds the ANALYZE threshold (calculated based on `autovacuum_analyze_threshold` + `autovacuum_analyze_scale_factor` x `reltuples`, where **reltuples** is the estimated number of rows in the table recorded in `pg_class`), **Analyze needed** is returned.
- If the number of modified rows in the table does not exceed the threshold of **analyze**, the message **Analyze not needed** is returned.

Return type: text

pg_stat_get_runtime_relstats(oid)

Description: Returns memory table-level statistics generated by lightweight autoanalyze.

Return type: SETOF record

pg_stat_get_runtime_attstats(oid)

Description: Returns memory table- and column- level statistics generated by lightweight autoanalyze.

Return type: SETOF record

pg_stat_get_runtime_partstats(oid)

Description: Returns memory partition level statistics generated by lightweight autoanalyze. Only 8.2.1.200 and later cluster versions support this function.

Return type: SETOF record

pg_stat_get_runtime_part_attstats(oid)

Description: Returns memory partition-column level statistics generated by lightweight autoanalyze. Only 8.2.1.200 and later cluster versions support this function.

Return type: SETOF record

pg_stat_get_runtime_exprstats(oid)

Description: Returns memory expression statistics generated by lightweight autoanalyze. Only 8.2.1.200 and later cluster versions support this function.

Return type: SETOF record

pg_stat_get_predicate_columns(oid)

Description: This function returns the predicate column information collected by a table. Only clusters of 9.1.0.100 and later versions support this function.

Return type: setof smallint

pg_stat_delete_predicate_columns(oid)

Description: This function clears the predicate column information of a table. Only clusters of 9.1.0.100 and later versions support this function.

Return type: void

pg_total_autovac_tuples(bool)

Description: Gets the tuple records related to **total autovac**, such as **nodename**, **nspname**, **relname**, and the IUD information of tuples.

Return type: SETOF record

pg_autovac_status(oid)

Description: Returns autovac information, such as **nodename**, **nspname**, **relname**, **analyze**, **vacuum**, thresholds of **analyze** and **vacuum**, and the number of analyzed or vacuumed tuples.

Return type: SETOF record

pg_autovac_timeout(oid)

Description: Returns the number of consecutive timeouts during the autovac operation on a table. If the table information is invalid or the node information is abnormal, **NULL** will be returned.

Return type: bigint

pg_autovac_coordinator(oid)

Description: Returns the name of the CN performing the autovac operation on a table. If the table information is invalid or the node information is abnormal, **NULL** will be returned.

Return type: text

pg_stat_get_all(regclass)

Description: This function retrieves a tuple record from the **pg_stat_object** table on the current CN. The record includes 29 columns, ranging from **databaseid** to **autovac_vacuum_count**. The input parameter is the table OID. Only 8.2.1.210 and later cluster versions support this function.

Return type: SETOF record

Example: Query the last access time of the **t1** table in the public schema on the current node.

```
SELECT data_access_timestamp FROM pg_stat_get_all('public.t1'::regclass);
data_access_timestamp
-----
2024-06-27 16:16:26.643132+08
(1 row)
```

pgxc_stat_get_all(relnamespace text, relname text)

Description: This function retrieves tuple records from the **pg_stat_object** table on all CNs. The record includes 29 columns, ranging from **databaseid** to **autovac_vacuum_count**. Only 8.2.1.210 and later cluster versions support this function.

Return type: SETOF record

Example: Query the last access time of the **t1** table in the public schema on all CNs.

```
SELECT relid,data_access_timestamp FROM pgxc_stat_get_all('public','t1');
relid  | data_access_timestamp
-----+-----
2147516676 | 2000-01-01 08:00:00+08
2147516674 | 2000-01-01 08:00:00+08
2147516655 | 2024-06-27 16:16:26.643132+08
(3 rows)
```

pgxc_stat_get_last_data_access_timestamp(relnamespace text, relname text)

Description: This function returns the last access time of a specified table on all nodes. Clusters of 9.1.0.100 and later versions support this function.

Return type: timestamptz

Example: Query the last access time of table **t1** in the public schema on all nodes.

```
SELECT * FROM pgxc_stat_get_last_data_access_timestamp('public','t1');
pgxc_stat_get_last_data_access_timestamp
-----
2024-06-27 16:16:26.643132+08
(1 row)
```

pgxc_stat_flush_object_data()

Description: This function enables manual persistence, where each node writes statistics on all tables in local databases to the **pg_stat_object** system catalog. The statistics include 29 columns from **databaseid** to **autovac_vacuum_count** in **pg_stat_object**. Only the system administrator can execute this function. Clusters of 9.1.0.100 and later versions support this function.

Return type: Boolean

Example: The system administrator performs persistence.

```
SELECT pgxc_stat_flush_object_data();
pgxc_stat_flush_object_data
-----
t
(1 row)
```

NOTE

This function cannot be executed concurrently.

pgxc_get_wlm_session_info_bytime(text, timestamp without time zone, timestamp without time zone, int)

Description: The query performance of the PGXC_WLM_SESSION_INFO view is poor if the view contains a large number of records. In this case, you are advised to use this function to filter the query. The input parameters are *time column* (**start_time** or **finish_time**), *start time*, *end time*, and *maximum number of records returned for each CN*. The return result is a subset of records in the **GS_WLM_SESSION_HISTORY** view.

Return type: SETOF record

pgxc_get_wlm_current_instance_info(text, int default null)

Description: Queries the current resource usage of each node in the cluster on the CN and reads the data that is not stored in the GS_WLM_INSTANCE_HISTORY system catalog in the memory. The input parameters are the node name (**ALL**, **C**, **D**, or *instance name*) and the maximum number of records returned by each node. The returned value is **GS_WLM_INSTANCE_HISTORY**.

Return type: SETOF record

pgxc_get_wlm_history_instance_info(text, TIMESTAMP, TIMESTAMP, int default null)

Description: Queries the historical resource usage of each cluster node on the CN node and reads data from the **GS_WLM_INSTANCE_HISTORY** system catalog. The input parameters are as follows: node name (**ALL**, **C**, **D**, or *instance name*), start time, end time, and maximum number of records returned for each instance. The returned value is **GS_WLM_INSTANCE_HISTORY**.

Return type: SETOF record

pg_stat_get_last_data_changed_time(oid)

Description: Returns the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE/TRUNCATE/DROP PARTITION** was performed last time on a table. The data in the **last_data_changed** column of the PG_STAT_ALL_TABLES view is calculated by using this function. The performance of obtaining the last modification time by using the view is poor when the table has a large amount of data. In this case, you are advised to use the function.

Return type: timestampz

pg_stat_set_last_data_changed_time(oid)

Description: Manually changes the time when **INSERT**, **UPDATE**, **DELETE**, or **EXCHANGE/TRUNCATE/DROP PARTITION** was performed last time.

Return type: void

pg_stat_set_last_data_changed_num(oid)

Description: Sets the history modification times of a table node.

Return type: void

pv_session_time()

Description: Collects statistics on the running time of each session thread on the current node and the time consumed in each execution phase.

Return type: record

pv_instance_time()

Description: Collects statistics on the running time of the current node and the time consumed in each execution phase.

Return type: record

pg_stat_get_activity(integer)

Description: Returns a record about the backend with the specified PID. A record for each active backend in the system is returned if **NULL** is specified. The return result is a subset of records (excluding the **connection_info** column) in the PG_STAT_ACTIVITY view.

Return type: SETOF record

pg_stat_get_activity_with_conninfo(integer)

Description: Returns a record about the backend with the specified PID. A record for each active backend in the system is returned if **NULL** is specified. The return result is a subset of records in the **PG_STAT_ACTIVITY** view.

Return type: SETOF record

pg_user_iostat(text)

Description: This function has been deprecated in version 8.1.2 and is invalid when used in queries in the current version. In cluster versions 8.2.0 and later, you can use the **PGXC_TOTAL_USER_RESOURCE_INFO** view to query the real-time resource consumption of users on instances.

Return type: record

| Name | Type | Description |
|---------------|------|---|
| userid | oid | User ID. |
| min_curr_iops | int4 | Minimum I/O of the current user across DNs. |
| max_curr_iops | int4 | Maximum I/O of the current user across DNs. |
| min_peak_iops | int4 | Minimum peak I/O of the current user across DNs. |
| max_peak_iops | int4 | Maximum peak I/O of the current user across DNs. |
| io_limits | int4 | io_limits set for the resource pool specified by the user. |
| io_priority | text | io_priority set for the user. |

pg_stat_get_function_calls(oid)

Description: Number of times the function has been called

Return type: bigint

pg_stat_get_function_total_time(oid)

Description: Gets the total wall-clock time spent on a function, in microseconds. The time spent on calling this function is included.

Return type: double precision

pg_stat_get_function_self_time(oid)

Description: Gets the time spent only on this function in the current transaction. The time spent on calling this function is not included.

Return type: double precision

pg_stat_get_backend_idset()

Description: Set of currently active server process numbers (from 1 to the number of active server processes)

Return type: setofinteger

pg_stat_get_backend_pid(integer)

Description: Thread ID of the given server thread

Return type: bigint

```
SELECT pg_stat_get_backend_pid(1);
pg_stat_get_backend_pid
-----
139706243217168
(1 row)
```

pg_stat_get_backend_dbid(integer)

Description: ID of the database connected to the given server process

Return type: OID

pg_stat_get_backend_userid(integer)

Description: User ID of the given server process

Return type: OID

pg_stat_get_backend_activity(integer)

Description: Active command of the given server process, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is on

Return type: text

pg_stat_get_backend_waiting(integer)

Description: True if the given server process is waiting for a lock, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is on

Return type: boolean

pg_stat_get_backend_activity_start(integer)

Description: The time at which the given server process's currently executing query was started, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is on

Return type: timestamp with time zone

pg_stat_get_backend_xact_start(integer)

Description: The time at which the given server process's currently executing transaction was started, but only if the current user is a system administrator or the same user as that of the session being queried and **track_activities** is on

Return type: timestamp with time zone

pg_stat_get_backend_start(integer)

Description: The time at which the given server process was started, or **NULL** if the current user is neither a system administrator nor the same user as that of the session being queried

Return type: timestamp with time zone

pg_stat_get_backend_client_addr(integer)

Description: IP address of the client connected to the given server process.

If the connection is over a Unix domain socket, or if the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** will be returned.

Return type: inet

Note: An IP address used as an input parameter of this function cannot contain periods (.). For example, **192.168.100.128** should be written as **192168100128**.

pg_stat_get_backend_client_port(integer)

Description: TCP port number of the client connected to the given server process

If the connection is over a Unix domain socket, **-1** will be returned. If the current user is neither a system administrator nor the same user as that of the session being queried, **NULL** will be returned.

Return type: integer

pg_stat_get_bgwriter_timed_checkpoints()

Description: The number of times the background writer has started timed checkpoints (because the **checkpoint_timeout** time has expired)

Return type: bigint

pg_stat_get_bgwriter_requested_checkpoints()

Description: The number of times the background writer has started checkpoints based on requests from the backend because **checkpoint_segments** has been exceeded or the **CHECKPOINT** command has been executed

Return type: bigint

pg_stat_get_bgwriter_buf_written_checkpoints()

Description: The number of buffers written by the background writer during checkpoints

Return type: bigint

pg_stat_get_bgwriter_buf_written_clean()

Description: The number of buffers written by the background writer for routine cleaning of dirty pages

Return type: bigint

pg_stat_get_bgwriter_maxwritten_clean()

Description: The number of times the background writer has stopped its cleaning scan because it has written more buffers than specified in the **bgwriter_lru_maxpages** parameter

Return type: bigint

pg_stat_get_buf_written_backend()

Description: The number of buffers written by the backend because they needed to allocate a new buffer

Return type: bigint

pg_stat_get_buf_alloc()

Description: The total number of buffer allocations

Return type: bigint

pg_stat_clear_snapshot()

Description: Discards the current statistics snapshot.

Return type: void

pg_stat_reset()

Description: Resets all statistics counters for the current database to zero (requires system administrator permissions).

Return type: void

pg_stat_reset_shared(text)

Description: Resets all statistics counters for the current database in each node in a shared cluster to zero (requires system administrator permissions).

Return type: void

pg_stat_reset_single_table_counters(oid)

Description: Resets statistics for a single table or index in the current database to zero (requires system administrator permissions).

Return type: void

pg_stat_reset_single_function_counters(oid)

Description: Resets statistics for a single function in the current database to zero (requires system administrator permissions).

Return type: void

pg_stat_session_cu(int, int, int)

Description: Obtains the compression unit (CU) hit statistics of sessions running on the current node.

Return type: record

gs_get_stat_session_cu(text, int, int, int)

Description: Obtains the CU hit statistics of all sessions running in a cluster.

Return type: record

gs_get_stat_db_cu(text, text, bigint, bigint, bigint)

Description: Obtains the CU hit statistics of a database in a cluster.

Return type: record

pg_stat_get_cu_mem_hit(oid)

Description: Obtains the number of CU memory hits of a column storage table in the current database of the current node.

Return type: bigint

pg_stat_get_cu_hdd_sync(oid)

Description: Obtains the number of times CU is synchronously read from a disk by a column-store table in the current database of the current node.

Return type: bigint

pg_stat_get_cu_hdd_asyn(oid)

Description: Obtains the number of times CU is asynchronously read from a disk by a column-store table in the current database of the current node.

Return type: bigint

pg_stat_get_db_cu_mem_hit(oid)

Description: Obtains the CU memory hit in a database of the current node.

Return type: bigint

pg_stat_get_db_cu_hdd_sync(oid)

Description: Obtains the times CU is synchronously read from a disk by a database of the current node.

Return type: bigint

pg_stat_get_db_cu_hdd_asyn(oid)

Description: Obtains the times CU is asynchronously read from a disk by a database of the current node.

Return type: bigint

pgxc_fenced_udf_process()

Description: Shows the number of UDF Master and Work processes.

Return type: record

pgxc_terminate_all_fenced_udf_process()

Description: Kills all UDF Work processes.

Return type: bool

gs_all_nodegroup_control_group_info(text)

Description: Provides Cgroup information for all logical clusters. Before invoking this function, you need to specify the name of a logical cluster to be queried. For example, to query the Cgroup information of the **installation** logical cluster, run the following statement:

```
SELECT * FROM gs_all_nodegroup_control_group_info('installation');
```

Return type: record

The following table describes return columns.

| Name | Type | Description |
|----------|--------|--|
| name | text | Name of a Cgroup |
| type | text | Type of the Cgroup |
| gid | bigint | Cgroup ID |
| classgid | bigint | ID of the Class Cgroup where a Workload Cgroup belongs |
| class | text | Class Cgroup |

| Name | Type | Description |
|----------|--------|-------------------------------------|
| workload | text | Workload Cgroup |
| shares | bigint | CPU quota allocated to a Cgroup |
| limits | bigint | Limit of CPUs allocated to a Cgroup |
| wdlevel | bigint | Workload Cgroup level |
| cpucores | text | Usage of CPU cores in a Cgroup |

gs_get_nodegroup_tablecount(name)

Description: Total number of user tables in all the databases in a logical cluster

Return type: integer

pgxc_max_datanode_size(name)

Description: Maximum disk space occupied by database files in all the DNs of a logical cluster. The unit is byte.

Return type: bigint

gs_check_logic_cluster_consistency()

Description: Checks whether the system information of all logical clusters in the system is consistent. If no record is returned, the information is consistent. Otherwise, the Node Group information on CNs and DNs in the logical cluster is inconsistent. This function cannot be invoked during redistribution in a scale-in or scale-out.

Return type: record

gs_check_tables_distribution()

Description: Checks whether the user table distribution in the system is consistent. If no record is returned, table distribution is consistent. This function cannot be invoked during redistribution in a scale-in or scale-out.

Return type: record

pg_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

Description: Obtains damage information about pages or CUs after the current node is started.

Return type: record

pgxc_stat_bad_block(text, int, int, int, int, int, timestamp with time zone, timestamp with time zone)

Description: Obtains damage information about pages or CUs after all the nodes in the cluster are started.

Return type: record

pg_stat_bad_block_clear()

Description: Deletes the page and CU damage information that is read and recorded on the node. (System administrator rights are required.)

Return type: void

pgxc_stat_bad_block_clear()

Description: Deletes the page and CU damage information that is read and recorded on all the nodes in the cluster. (System administrator rights are required.)

Return type: void

gs_respool_exception_info(pool text)

Description: Queries for the query rule of a specified resource pool.

Return type: record

gs_control_group_info(pool text)

Description: Queries for information about Cgroups associated with a resource pool.

Return type: record

The following information is displayed:

| Attribute | Value | Description |
|-----------|---------------------|---|
| name | class_a:workload_a1 | Class name and workload name |
| class | class_a | Class Cgroup name |
| workload | workload_a1 | Workload Cgroup name |
| type | DEFWD | Cgroup type (Top, CLASS, BAKWD, DEFWD, and TSWD) |
| gid | 87 | Cgroup ID |
| shares | 30 | Percentage of CPU resources to those on the parent node |
| limits | 0 | Percentage of CPU cores to those on the parent node |
| rate | 0 | Allocation ratio in Timeshare |

| Attribute | Value | Description |
|-----------|-------|---------------------|
| cpucores | 0-3 | Number of CPU cores |

gs_wlm_user_resource_info(name text)

Description: Queries for a user's resource quota and resource usage.

Return type: record

pgxc_stat_single_table(schema, tablename)

Description: Executed on CNs, with the schema name and table name passed. This function queries the statistics of a single table in the entire database and the dirty page rate of the table on each DN.

This function is supported only by version 8.1.3 or later clusters.

NOTE

The statistics of this function depend on the **ANALYZE** operation. To obtain the most accurate information, perform the **ANALYZE** operation on the table first.

Return type: record

The return value fields are the same as those of the [pg_stat_get_tuple\(\)](#) function.

```
SELECT * FROM pgxc_stat_single_table('public','t1');
 nodeid | tableid | partid | last_vacuum | last_autovacuum | last_analyze |
 last_autoanalyze | vacuum_count | autovacuum_count | analyze_count | autoanalyze_count | n_tup_ins | n_
 tup_upd | n_tup_del | n_tup_hot_upd | n_tup_change | n_live_tup | n_dead_tup | dirty_rate |
 last_data_changed
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 datanode1 | 1270075 |      | 2000-01-01 08:00:00+08 | 2000-01-01 08:00:00+08 | 2023-01-09
 09:38:43.220876+08 | 2000-01-01 08:00:00+08 |      0 |      0 |      1 |      0 |      0 |
      0 |      0 |      0 |      0 |      0 |      0 |
(1 row)
```

pgxc_stat_wal_write()

Description: Records information about DN threads that interact with CNs, and collects statistics on WAL logs and data page import volume and rate. This function is executed on all CNs. This parameter is supported by version 8.2.0 or later clusters.

Return type: bigint

Fields in the returned value:

- **node_name**: instance name.
- **application_name**: application name.
- **query_start**: start time of the statement that is being executed.
- **datapage_write**: total size of data pages generated by the current query, in bytes

- **datapage_write_speed**: data page import speed, in bytes/s.
- **wal_write**: total size WAL logs imported by the current query, in bytes.
- **wal_write_speed**: WAL log import speed, in bytes/s.
- **total_datapage_write**: total size of data pages generated by the current thread, in bytes.
- **total_wal_write**: total size of WAL logs, in bytes.

```
SELECT * FROM pgxc_stat_wal_write();
node_name | datid | pid | lwtid | usesysid | application_name | state | query_start
| backend_start | query_id | datapage_write | datapage_write_speed | wal_write |
wal_write_speed | total_datapage_write | total_wal_write
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
datanode1 | 15979 | 140328473442048 | 12408 | 10 | CalculateSpaceInfo | active | 2022-11-15
10:39:45.00219+08 | 2022-11-15 10:39:45.000918+08 | 0 | 0 | 0 | 70697
| 0 | 0 | 0
datanode1 | 0 | 140328075503360 | 12510 | 10 | WalSender to Secondary | idle
| 2022-11-15 10:39:46.708557+08 | 0 | 0 | 210
| 0 | 0 | 0
datanode1 | 15979 | 140327896741632 | 13612 | 10 | coordinator1 | active | 2022-11-15
10:46:32.832548+08 | 2022-11-15 10:40:20.117516+08 | 72620543991349940 | 586579968 |
17425000 | 11743056 | 348000 | 5337505792 | 107245825
datanode1 | 15979 | 140327583217408 | 13614 | 10 | coordinator1 | active | 2022-11-15
10:46:32.832548+08 | 72620543991349940 | 0 | 0 | 0
| 0 | 0 | 0
datanode1 | 15979 | 140327485175552 | 27914 | 10 | coordinator1 | active | 2022-11-15
10:47:06.493584+08 | 2022-11-15 10:47:06.489062+08 | 72620543991350020 | 0 | 0
| 0 | 0 | 8675
(5 rows)
```

NOTE

When row-store data is imported in batches without indexes, the Xlogs related to logical new pages are generated during data page copy. If the volume of Xlogs is greater than the default value, flow control will be triggered.

pg_stat_wal_write()

Description: This function is executed on a CN or DN to record thread information on it and collect statistics on WAL logs and the volume and rate of imported data pages. The following uses a CN as an example.

This parameter is supported by version 8.2.0 or later clusters.

Return type: bigint

Fields in the returned value:

- **node_name**: instance name.
- **application_name**: application name.
- **query_start**: start time of the statement that is being executed.
- **datapage_write**: total size of data pages generated by the current query, in bytes
- **datapage_write_speed**: data page import speed, in bytes/s.
- **wal_write**: total size WAL logs imported by the current query, in bytes.
- **wal_write_speed**: WAL log import speed, in bytes/s.
- **total_datapage_write**: total size of data pages generated by the current thread, in bytes.

- **total_wal_write**: total size of WAL logs, in bytes.

```
SELECT * FROM pg_stat_wal_write();
```

| node_name | datid | pid | lwtid | usesysid | application_name | state | query_start | backend_start | query_id | datapage_write | datapage_write_speed | wal_write | wal_write_speed | total_datapage_write | total_wal_write |
|--------------|-------|-----------------|-------|----------|--------------------|--------|-------------------------------|-------------------------------|-------------------|----------------|----------------------|-----------|-----------------|----------------------|-----------------|
| coordinator1 | 15979 | 140234153498368 | 25353 | 10 | JobScheduler | active | 2022-11-15 11:57:54.396347+08 | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140234132027136 | 25354 | 10 | StatCollector | idle | | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140234090084096 | 25356 | 10 | WDRSnapshot | idle | 2022-11-15 11:57:54.388948+08 | 2022-11-15 11:57:54.403055+08 | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140234027169536 | 25359 | 10 | workload | active | 2022-11-15 11:57:54.404836+08 | 2022-11-15 11:57:54.389259+08 | 72620543991404472 | | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140234004621056 | 25360 | 10 | CalculateSpaceInfo | active | 2022-11-15 11:57:54.405677+08 | 2022-11-15 11:57:54.389847+08 | 72620543991349266 | | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140233987839744 | 25361 | 10 | WorkloadMonitor | active | 2022-11-15 11:57:54.406263+08 | 2022-11-15 11:57:54.390373+08 | 72620543991404418 | | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140233971058432 | 25362 | 10 | WLMArbitrer | active | 2022-11-15 11:57:54.406367+08 | 2022-11-15 11:57:54.390947+08 | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140233228666624 | 14415 | 10 | gsq | active | 2022-11-15 21:19:32.200305+08 | 2022-11-15 21:09:38.916931+08 | 72620543991404422 | | 0 | 0 | 0 | | |
| coordinator1 | 15979 | 140233172035328 | 15516 | 10 | gsq | active | 2022-11-15 21:19:49.7877+08 | 2022-11-15 21:10:13.447312+08 | 72620543991404485 | | 0 | 0 | 0 | | |
| coordinator1 | 0 | 140234245269248 | 25348 | 0 | Background writer | idle | 2022-11-15 11:57:54.383651+08 | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 0 | 140234291410688 | 25347 | 0 | CheckPointer | idle | 2022-11-15 11:57:54.383231+08 | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 0 | 140234228487936 | 25349 | 0 | Wal Writer | idle | 2022-11-15 11:57:54.384069+08 | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 0 | 140234068584192 | 25357 | 0 | TwoPhase Cleaner | idle | 2022-11-15 11:57:54.388332+08 | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 0 | 140234048657152 | 25358 | 0 | LWLock Monitor | idle | 2022-11-15 11:57:54.389239+08 | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 0 | 140234194925312 | 25351 | 0 | CBM Writer | idle | 2022-11-15 11:57:54.38506+08 | | | 0 | 0 | 0 | 0 | | |
| coordinator1 | 0 | 140234211706624 | 25350 | 0 | Tick service | idle | 2022-11-15 11:57:54.384136+08 | | | 0 | 0 | 0 | 0 | | |

(16 rows)

NOTE

When row-store data is imported in batches without indexes, the Xlogs related to logical new pages are generated during data page copy. If the volume of Xlogs is greater than the default value, flow control will be triggered.

gs_stack()

Description: Obtains the stack information of CN or DN processes. The input parameter **tid** needs to be obtained from views. **lwtid** is not supported. This function is supported by version 8.2.1 or later clusters.

gs_stack() provides a more flexible and quicker method to obtain the stack than **gdb** and **gstack** commands. It is recommended for administrators to use during problem analysis and locating. It is not recommended that this tool be used as a routine monitoring tool.

Fields in the returned value:

- **tid**: thread ID.
- **lwtid**: lightweight thread ID.
- **stack**: stack information corresponding to the **tid** or **lwtid** thread.

The **gs_stack()** function can be used in the following four ways:

- Method 1: If the input parameter value of the **gs_stack(tid)** function is 0, stacks of all threads in the CN or DN process connected to gsql are printed.

NOTE

- This statement can be executed by connecting to a CN or DN using gsql.
- The process of obtaining the running stacks of all threads in a CN or DN takes a long time, which affects normal service running. Therefore, you are advised not to run this command frequently.

```
SELECT * FROM gs_stack(0);
```

```
tid      | lwtid |
```

```
stack
```

```
-----+-----
```

```
+-----+-----
```

```
---
```

```
140186894757888 | 95275 | __poll +
```

```
0x2d
```

```
| | ServerLoop() +
```

```
0x457
```

```
| | PmStartupThreads() +
```

```
0x151
```

```
| | PostmasterMain(int, char**) +
```

```
0x22a
```

```
| | main +
```

```
0x250
```

```
| | __libc_start_main +
```

```
0xf5
```

```
| |
```

```
0xb8d2d7
```

```
| |
```

```
140185903825152 | 95316 | __poll +
```

```
0x2d
```

```
| | GsWaiter::wait(void*, long) +
```

```
0x11a
```

```
| | GsWaiter::wait(void*, int, int, long) +
```

```
0x7e
```

```
| | SysLoggerMain() +
```

```
0x109
```

```
| | SubPostmasterMain(tag_gs_thread_args*) +
```

```
0xc9f
```

```
| | MainStarterThreadFunc(void*) +
```

```
0x6a
```

```
| | ThreadStarterFunc(void*) +
```

```
0x66
```

```
| | start_thread +
```

```
0xc5 | | clone +
0x6d | | 140185903825528 | 95317 | __poll +
0x2d | | GsWaiter::wait(void*, long) +
0x11a | | AlarmCheckerMain() +
0x21b | | SubPostmasterMain(tag_gs_thread_args*) +
0xcc6 | | MainStarterThreadFunc(void*) +
0x6a | | ThreadStarterFunc(void*) +
0x66 | | start_thread +
0xc5 | | clone +
0x6d | | 140185903825904 | 95320 | __poll +
0x2d | | GsWaiter::wait(void*, long) +
0x11a | | ReaperBackendMain() +
0x1ee | | SubPostmasterMain(tag_gs_thread_args*) +
0xd6f | | MainStarterThreadFunc(void*) +
0x6a | | ThreadStarterFunc(void*) +
0x66 | | start_thread +
0xc5 | | clone +
0x6d | | 140185903826656 | 95324 | __poll +
0x2d | | GsWaiter::wait(void*, long) +
0x11a | | CheckpointerMain() +
0x8bb | | AuxiliaryThreadUnderPm(AuxProcType) +
0x210 | | SubPostmasterMain(tag_gs_thread_args*) +
0x7e0 | | MainStarterThreadFunc(void*) +
0x6a | | ThreadStarterFunc(void*) +
0x66 | | start_thread +
0xc5 | | clone +
0x6d | | 140185903827032 | 95325 | __poll +
0x2d | | GsWaiter::wait(void*, long) +
0x11a | | BackgroundWriterMain() +
0x541 | | AuxiliaryThreadUnderPm(AuxProcType) +
0x210 | | SubPostmasterMain(tag_gs_thread_args*) +
0x7e0 | | MainStarterThreadFunc(void*) +
0x6a |
```

```
0x66 | | ThreadStarterFunc(void*) +
```

- Method 2: If the input parameter of the **gs_stack(tid)** function is set to a specified tid, the stack information of the tid thread in the CN or DN process connected using gsql is printed.

NOTE

- This statement can be executed by connecting to a CN or DN using gsql.
- tid** must be the ID of an existing thread on the connected CN or DN process. Otherwise, the error message "invalid thread id" is displayed.
- tid** is obtained from other views. **lwtid** is not supported.
- The process of obtaining the running stack of a specified thread in a specified CN or DN process takes a short time, which does not affect services. Therefore, this method is recommended.

```
SELECT * FROM gs_stack(140185903826656);
tid      | lwtid | stack
-----+-----
140185903826656 | 95275 | __poll + 0x2d +
          |      | GsWaiter::wait(void*, long) + 0x11a +
          |      | CheckpointerMain() + 0x8bb +
          |      | AuxiliaryThreadUnderPm(AuxProcType) + 0x210 +
          |      | SubPostmasterMain(tag_gs_thread_args*) + 0x7e0+
          |      | MainStarterThreadFunc(void*) + 0x6a +
          |      | ThreadStarterFunc(void*) + 0x66 +
          |      | start_thread + 0xc5 +
          |      | clone + 0x6d +
(1 row)
```

- Method 3: If the **gs_stack()** function is set to **gs_stack('nodename', 0)**, stacks of all threads in the process specified by **nodename** are printed.

NOTE

- The underlying implementation uses **execute direct on**. Therefore, you must use gsql to connect to the CN to run the command.
- The first parameter **nodename** must be enclosed in single quotation marks.
- The process of obtaining the running stacks of all threads in a CN or DN takes a long time, which affects normal service running. Therefore, you are advised not to run this command frequently.

```
SELECT * FROM gs_stack('datanode2', 0);
tid      | lwtid | stack
-----+-----
140634541242112 | 95442 | do_futex_wait +
0x5f      |      | +
          |      | __new_sem_wait_slow +
0x57      |      | +
          |      | sem_timedwait +
0x35      |      | +
          |      | tagBinarySemaphore::timed_wait(int) +
0x4f      |      | +
          |      | AuxiliaryThreadMain(void*) +
0x6d      |      | +
          |      | LibcommThreadRoutine(void*) +
0x41      |      | +
          |      | start_thread +
0xc5      |      | +
          |      | clone +
0x6d      |      | +
```

```
140634549634816 | 95438 | epoll_wait +
0x33      |      |      +
0x38      |      | LibcommEpollWait(int, int) +
0x8e      |      | RecvDataThreadMain(void*) +
0x41      |      | LibcommThreadRoutine(void*) +
0xc5      |      | start_thread +
0x6d      |      | clone +
140634558027520 | 95432 | epoll_wait +
0x33      |      |      +
0x38      |      | LibcommEpollWait(int, int) +
0x8e      |      | RecvDataThreadMain(void*) +
0x41      |      | LibcommThreadRoutine(void*) +
0xc5      |      | start_thread +
0x6d      |      | clone +
140634566420224 | 95430 | epoll_wait +
0x33      |      |      +
0x38      |      | LibcommEpollWait(int, int) +
0x8e      |      | RecvDataThreadMain(void*) +
0x41      |      | LibcommThreadRoutine(void*) +
0xc5      |      | start_thread +
0x6d      |      | clone +
140634589476608 | 95426 | epoll_wait +
0x33      |      |      +
0x38      |      | LibcommEpollWait(int, int) +
0x8e      |      | RecvDataThreadMain(void*) +
0x41      |      | LibcommThreadRoutine(void*) +
0xc5      |      | start_thread +
0x6d      |      | clone +
140634597869312 | 95424 | do_futex_wait +
0x5f      |      |      +
0x57      |      | __new_sem_wait_slow +
0x35      |      | sem_timedwait +
0x4f      |      | tagBinarySemaphore::timed_wait(int) +
0x58      |      | SendDataWait() +
0xe4      |      | SendDataThreadMain(void*) +
0x41      |      | LibcommThreadRoutine(void*) +
0xc5      |      | start_thread +
0x6d      |      | clone +
```

```
0x6d | | | +
140634606262016 | 95387 | do_futex_wait +
0x5f | | | +
0x57 | | | +
0x35 | | | +
0x4f | | | +
0x58 | | | +
0xe4 | | | +
```

- Method 4: If the **gs_stack()** function is set to **gs_stack('nodename', tid)**, stack information of the **tid** thread in the process specified by **nodename** are printed.

NOTE

- The underlying implementation uses **execute direct on**. Therefore, you must use **gsql** to connect to the CN to run the command.
- The first parameter **nodename** must be enclosed in single quotation marks.
- The second parameter must be a thread that exists in the process specified by **nodename**. Otherwise, the error message "invalid thread id" is displayed.
- **tid** is obtained from other views. **lwtid** is not supported.
- The process of obtaining the running stack of a specified thread in a specified CN or DN process takes a short time, which does not affect services. Therefore, this method is recommended.

```
SELECT * FROM gs_stack('datanode2', 140634549634816);
tid | lwtid | stack
-----+-----+-----
140634549634816 | 95438 | epoll_wait + 0x33 +
| | LibcommEpollWait(int, int) + 0x38 +
| | RecvDataThreadMain(void*) + 0x8e +
| | LibcommThreadRoutine(void*) + 0x41+
| | start_thread + 0xc5 +
| | clone + 0x6d +
(1 row)
```

pgxc_get_xlog_stats()

Description: Collects statistics on the number of Xlogs of different types from the time when each DN is started to the current time. This command is executed on CNs. This function is supported only by clusters of version 8.3.0 or later.

Return type: record

The following information is displayed:

| Name | Type | Description |
|--------------|--------|----------------|
| node_name | text | Node name |
| rmgr_name | text | Xlog main type |
| record_info | text | xlog info |
| record_count | bigint | Quantity |

| Name | Type | Description |
|---------------|--------|---------------------------------------|
| count_pct | text | Count (%) |
| record_size | bigint | Record size, in bytes. |
| record_pct | text | Percentage of the record size. |
| fpi_size | bigint | Full page image (FPI) size, in bytes. |
| fpi_pct | text | Percentage of the FPI size. |
| combined_size | bigint | Total size of Xlogs, in bytes. |
| combined_pct | text | Percentage of the total Xlog size. |

 NOTE

- This function collects statistics only on the number and size of Xlogs of different types since the last DN startup.
- The xlog size calculated by this function does not consider the xlog spilling. As a result, the xlog size is different from the actual xlog file size.

```
SELECT * FROM pgxc_get_xlog_stats();
```

| node_name | rmgr_name | record_info | record_count | count_pct | record_size | record_pct | fpi_size | fpi_pct | combined_size | combined_pct |
|-----------|-------------|----------------------|--------------|-----------|-------------|------------|----------|---------|---------------|--------------|
| datanode1 | XLOG | checkpoint: shutdown | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | checkpoint: online | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | xlog no-op | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | nextOid | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | xlog switch | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | backup end | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | parameter change | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | restore point | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | full page writes | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | xlog recovery end | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | XLOG | page hint | 1 | 0.12% | 56 | 0.11% | 8192 | 17.92% | 8248 | 8.71% |
| datanode1 | XLOG | xlog fpi | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | Transaction | commit | 3 | 0.35% | 246 | 0.50% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | Transaction | prepare | 1 | 0.12% | 469 | 0.96% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | Transaction | abort | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% | 0 | 0.00% |
| datanode1 | Transaction | commit prepared | 1 | 0.12% | 106 | 0.22% | 0 | 0.00% | 0 | 0.00% |

| | | | | | |
|-----------|---|--|--------------|----------------|-----------|
| 0.00% | 106 0.11% | | | | |
| datanode1 | Transaction abort prepared | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Transaction xid assignment xtop | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Transaction commit compact | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Storage file create | | 0 0.00% | 0 0.00% | 0 0.00% |
| | 0 0.00% | | | | |
| datanode1 | Storage file truncate | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Storage vacuum clear file | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | CLOG zero page | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | CLOG truncate before | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Database create db | | 1 0.12% | 50 0.10% | 0 |
| 0.00% | 50 0.05% | | | | |
| datanode1 | Database drop db | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Tablespace create tablespace | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Tablespace drop tablespace | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Tablespace relative create | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | MultiXact zero offsets page | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | MultiXact zero members page | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | MultiXact create multixact | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | RelMap update relmap | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Standby AccessExclusive locks | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Standby xlog running xacts | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Standby release AccessExclusive locks | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Standby xlog standby csn | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap2 freeze | | 0 0.00% | 0 0.00% | 0 0.00% |
| | 0 0.00% | | | | |
| datanode1 | Heap2 clean | | 0 0.00% | 0 0.00% | 0 0.00% |
| | 0 0.00% | | | | |
| datanode1 | Heap2 page upgrade | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap2 cleanup info | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap2 visible | | 0 0.00% | 0 0.00% | 0 0.00% |
| | 0 0.00% | | | | |
| datanode1 | Heap2 multi-insert | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap2 bcm | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap2 logical newpage rel | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap2 no repair page | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap2 multi-insert(init) base xid | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |
| datanode1 | Heap insert | | 280 32.67% | 15964 32.63% | 14314 |
| 31.31% | 30278 31.99% | | | | |
| datanode1 | Heap delete | | 0 0.00% | 0 0.00% | 0 0.00% |
| | 0 0.00% | | | | |
| datanode1 | Heap update | | 0 0.00% | 0 0.00% | 0 |
| 0.00% | 0 0.00% | | | | |

| | | | | |
|-------------------------|-----------------------|--------------|----------------|-------------|
| datanode1 Heap | base shift | 0 0.00% | 0 0.00% | 0 0.00% |
| 0 0.00% | | | | |
| datanode1 Heap | hot update | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Heap | new page | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Heap | lock type | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Heap | inplace | 2 0.23% | 116 0.24% | 8164 |
| 17.86% 8280 8.75% | | | | |
| datanode1 Heap | init page | 2 0.23% | 130 0.27% | 92 |
| 0.20% 222 0.23% | | | | |
| datanode1 Heap | insert(init) base xid | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Btree | insert leaf | 562 65.58% | 31484 64.35% | 13352 |
| 29.21% 44836 47.37% | | | | |
| datanode1 Btree | insert upper | 1 0.12% | 72 0.15% | 148 |
| 0.32% 220 0.23% | | | | |
| datanode1 Btree | insert meta | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Btree | split left | 1 0.12% | 78 0.16% | 744 1.63% |
| 822 0.87% | | | | |
| datanode1 Btree | split right | 0 0.00% | 0 0.00% | 0 0.00% |
| 0 0.00% | | | | |
| datanode1 Btree | split left root | 0 0.00% | 0 0.00% | 0 0.00% |
| 0 0.00% | | | | |
| datanode1 Btree | split right root | 1 0.12% | 78 0.16% | 672 |
| 1.47% 750 0.79% | | | | |
| datanode1 Btree | delete | 0 0.00% | 0 0.00% | 0 0.00% |
| 0 0.00% | | | | |
| datanode1 Btree | delete page | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Btree | delete page meta | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Btree | lev | 1 0.12% | 78 0.16% | 40 0.09% |
| 118 0.12% | | | | |
| datanode1 Btree | delete page half | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Btree | vacuum | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Btree | reuse page | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Create index | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Create ptree | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Insert item | 0 0.00% | 0 0.00% | 0 0.00% |
| 0 0.00% | | | | |
| datanode1 Gin | Page split | 0 0.00% | 0 0.00% | 0 0.00% |
| 0 0.00% | | | | |
| datanode1 Gin | Vacuum page | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Delete page | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Update metapage | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Insert new list page | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Delete list pages | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gin | Vacuum data leaf page | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gist | page update | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gist | page split | 0 0.00% | 0 0.00% | 0 0.00% |
| 0 0.00% | | | | |
| datanode1 Gist | Create index | 0 0.00% | 0 0.00% | 0 |
| 0.00% 0 0.00% | | | | |
| datanode1 Gist | page delete | 0 0.00% | 0 0.00% | 0 |

[illegible]

pgxc_get_wal_speed()

Description: Obtains the WAL generation rate of each DN and the receive, write, flush, and redo rates of standby DNs. This function is supported only by clusters of version 8.3.0 or later.

Return type: record

The following information is displayed:

| Name | Type | Description |
|----------------|--------|---|
| node_name | text | Node name. |
| pid | bigint | PID of a thread. |
| local_role | text | Role of the current node |
| peer_role | text | Role of the recipient |
| peer_state | text | Status of the recipient |
| state | text | Transmission status |
| generate_speed | bigint | Generation rate of the primary DN wal, in byte/s. |

| Name | Type | Description |
|---------------|--------|---|
| receive_speed | bigint | Receiving rate of the standby DN. The unit is byte/s. |
| write_speed | bigint | Write speed of the standby DN. The unit is byte/s. |
| flush_speed | bigint | Disk flushing rate of the standby DN. The unit is byte/s. |
| redo_speed | bigint | Redo speed of the standby DN. The unit is byte/s. |

```
SELECT * FROM pgxc_get_wal_speed();
node_name | pid | local_role | peer_role | peer_state | state | generate_speed | receive_speed |
write_speed | flush_speed | apply_speed
-----+-----+-----+-----+-----+-----+-----+-----+
datanode1 | 140240389260824 | Primary | Secondary | Normal | Streaming | 0 | 0
| 0 | 0 | 0
datanode1 | 140240389262192 | Primary | Standby | Normal | Streaming | 0 | 0
| 0 | 0 | 0
(2 rows)
```

pg_xlog_display_one_lsn(start_lsn)

Description: This command can be executed on CNs or DNs to completely parse Xlogs in the current location based on the start LSN. This function is supported only by clusters of version 8.3.0 or later.

Input parameter: **start_lsn**, indicating the start LSN. Ensure that the entered start LSN is the beginning of an Xlog.

Return type: record

The following information is displayed:

| Name | Type | Description |
|-----------|------|---------------------------------|
| node_name | text | Current instance name. |
| start_lsn | text | Start LSN. |
| end_lsn | text | End LSN. |
| startlsn | text | Start LSN of the Xlog. |
| endlsn | text | End LSN of the Xlog. |
| prelsn | text | Start LSN of the previous Xlog. |

| Name | Type | Description |
|----------|------|---------------------------------------|
| xid | xid | Xlog transaction ID. |
| datalen | int4 | Xlog data length, in bytes. |
| totallen | int4 | Xlog length, in bytes. |
| type | text | Xlog type. |
| desc | text | Xlog content. |
| blkref | text | Relfilenode associated with the Xlog. |

Example:

```
SELECT * FROM pg_xlog_display_one_lsn('0/101CA00');
node_name | startlsn | endlsn | prelsn | xlog_tid | datalen | totallen | xlogtype |
xlogdescribe
          | blkref
-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----
datanode1 | 0/101CA00 | 0/101CA78 | 0/101C3F8 |      5 |    80 |   114 | Transaction | commit:
2023-10-19 22:21:38.617092+08; csn:0; inval msgs: catcache 11 ca
tcache 10 |
(1 row)
```

pg_xlogdump (tablename)

Description: Parses and filters Xlog files based on table names on CNs or DN. This function is supported only by clusters of version 8.3.0 or later.

Return type: record

The following information is displayed:

| Name | Type | Description |
|-----------|------|---------------------------------|
| node_name | text | Node name. |
| start_lsn | text | Start LSN. |
| end_lsn | text | End LSN. |
| startlsn | text | Start LSN of the Xlog. |
| endlsn | text | End LSN of the Xlog. |
| prelsn | text | Start LSN of the previous Xlog. |
| xid | xid | Xlog transaction ID. |

| Name | Type | Description |
|---------|------|---------------------------------------|
| datalen | int4 | Xlog data length, in bytes. |
| totalen | int4 | Xlog length, in bytes. |
| type | text | Xlog type. |
| desc | text | Xlog content. |
| blkref | text | Relfilenode associated with the Xlog. |

 NOTE

- The length of the **xlogdescribe** field varies depending on the Xlog type. The **pg_xlogdump()** function truncates the field and retains only the first 64 bytes.
- You can find the target Xlog and use the **pg_xlog_display_one_lsn()** function to view the complete Xlog content.

Example:

```
SELECT * FROM pg_xlogdump(pg_class);
node_name | startlsn | endlsn | prelsn | xlog_tid | datalen | totalen | xlogtype | xlogdescribe
|          |         |         |         |          |         |         |          |
+-----+-----+-----+-----+-----+-----+-----+-----+
datanode1 | 0/2DFC660 | 0/2DFE1D0 | 0/2DFA610 | 0 | 2 | 7000 | Heap | inplace: off 17 |
blkrel #0: rel 1663/16324/15920, fork main, blk 2
datanode1 | 0/2E02270 | 0/2E02D88 | 0/2E00220 | 0 | 2 | 2840 | Heap | inplace: off 2 | blkrel
#0: rel 1663/16324/15920, fork main, blk 0
datanode1 | 0/2E06E60 | 0/2E06F68 | 0/2E06E28 | 0 | 2 | 264 | Heap | inplace: off 10 | blkrel
#0: rel 1663/16324/15920, fork main, blk 0
datanode1 | 0/2E0FC08 | 0/2E0FD10 | 0/2E0EBB8 | 0 | 2 | 264 | Heap | inplace: off 13 | blkrel
#0: rel 1663/16324/15920, fork main, blk 0
datanode1 | 0/2E12960 | 0/2E12A68 | 0/2E11960 | 0 | 2 | 264 | Heap | inplace: off 15 | blkrel
#0: rel 1663/16324/15920, fork main, blk 0
datanode1 | 0/2E12C88 | 0/2E12D90 | 0/2E12B78 | 0 | 2 | 264 | Heap | inplace: off 17 | blkrel
#0: rel 1663/16324/15920, fork main, blk 0
datanode1 | 0/2E13FF8 | 0/2E14118 | 0/2E13110 | 0 | 2 | 264 | Heap | inplace: off 18 | blkrel
#0: rel 1663/16324/15920, fork main, blk 0
datanode1 | 0/2E169F8 | 0/2E185E0 | 0/2E169B0 | 0 | 2 | 7120 | Heap | inplace: off 1 | blkrel
#0: rel 1663/16324/15920, fork main, blk 1
datanode1 | 0/2E188B8 | 0/2E189D0 | 0/2E185E0 | 0 | 2 | 280 | Heap | inplace: off 2 | blkrel
#0: rel 1663/16324/15920, fork main, blk 1
datanode1 | 0/2E18BF8 | 0/2E18CE0 | 0/2E189D0 | 0 | 2 | 232 | Heap | inplace: off 28 | blkrel
#0: rel 1663/16324/15920, fork main, blk 1
datanode1 | 0/2E18E80 | 0/2E18F88 | 0/2E18DC0 | 0 | 2 | 264 | Heap | inplace: off 4 | blkrel
#0: rel 1663/16324/15920, fork main, blk 1
datanode1 | 0/2E19160 | 0/2E19268 | 0/2E19088 | 0 | 2 | 264 | Heap | inplace: off 5 | blkrel
#0: rel 1663/16324/15920, fork main, blk 1
datanode1 | 0/2E19460 | 0/2E1B1D0 | 0/2E19388 | 0 | 2 | 7508 | Heap | inplace: off 16 | blkrel
#0: rel 1663/16324/15920, fork main, blk 6
datanode1 | 0/2E1B1D0 | 0/2E1B2E8 | 0/2E19460 | 0 | 2 | 280 | Heap | inplace: off 18 | blkrel
#0: rel 1663/16324/15920, fork main, blk 6
datanode1 | 0/2E1B2E8 | 0/2E1B3D0 | 0/2E1B1D0 | 0 | 2 | 232 | Heap | inplace: off 30 | blkrel
#0: rel 1663/16324/15920, fork main, blk 1
(15 rows)
```

pg_xlogdump (xid)

Description: Parses and filters Xlogs based on transaction IDs. This statement can be executed on CNs or DN. This function is supported only by clusters of version 8.3.0 or later.

Return type: record

The following information is displayed:

| Name | Type | Description |
|-----------|------|---------------------------------------|
| node_name | text | Node name. |
| start_lsn | text | Start LSN. |
| end_lsn | text | End LSN. |
| startlsn | text | Start LSN of the Xlog. |
| endlsn | text | End LSN of the Xlog. |
| prelsn | text | Start LSN of the previous Xlog. |
| xid | xid | Xlog transaction ID. |
| datalen | int4 | Xlog data length, in bytes. |
| totalen | int4 | Xlog length, in bytes. |
| type | text | Xlog type. |
| desc | text | Xlog content. |
| blkref | text | Relfilenode associated with the Xlog. |

NOTE

- The length of the **xlogdescribe** field varies depending on the Xlog type. The **pg_xlogdump()** function truncates the field and retains only the first 64 bytes.
- You can find the target Xlog and use the **pg_xlog_display_one_lsn()** function to view the complete Xlog content.

Example:

```
SELECT * FROM pg_xlogdump(1);
node_name | startlsn | endlsn | prelsn | xlog_tid | datalen | totalen | xlogtype | xlogdescribe
|          |         |        |        |          |         |         |          |
-----+-----+-----+-----+-----+-----+-----+-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----
datanode1 | 0/10000A0 | 0/10020F0 | 0/1000028 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/1255, fork fsm, blk 2
datanode1 | 0/10020F0 | 0/1004140 | 0/10000A0 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/1247, fork fsm, blk 2
datanode1 | 0/1004140 | 0/1004168 | 0/10020F0 | 1 | 4 | 38 | XLOG | nextOid: 18192 |
datanode1 | 0/1004168 | 0/10061B8 | 0/1004140 | 1 | 0 | 8248 | XLOG | page hint | blkrel
```

```
#0: rel 1663/1/2616, fork fsm, blk 2
datanode1 | 0/10061B8 | 0/1008208 | 0/1004168 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/2602, fork fsm, blk 2
datanode1 | 0/1008208 | 0/100A258 | 0/10061B8 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/2603, fork fsm, blk 2
datanode1 | 0/100A258 | 0/100C2A8 | 0/1008208 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/2609, fork fsm, blk 2
datanode1 | 0/100C2A8 | 0/100E2F8 | 0/100A258 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/2605, fork fsm, blk 2
datanode1 | 0/100E2F8 | 0/1010348 | 0/100C2A8 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/1259, fork fsm, blk 2
datanode1 | 0/1010348 | 0/1012398 | 0/100E2F8 | 1 | 0 | 8248 | XLOG | page hint | blkrel
#0: rel 1663/1/2610, fork fsm, blk 2
(10 rows)
```

pg_xlogdump (start_lsn, end_lsn)

Description: Parses Xlog files based on the start and end LSNs. This command can be executed on CNs or DN. This function is supported only by clusters of version 8.3.0 or later.

Input parameters: **start_lsn** indicates the start LSN, and **end_lsn** indicates the end LSN. You do not need to ensure that the start LSN is the first Xlog. The parsing starts from the first valid Xlog after the start LSN.

Return type: record

The following information is displayed:

| Field | Type | Description |
|-----------|------|---------------------------------------|
| node_name | text | Node name. |
| start_lsn | text | Start LSN. |
| end_lsn | text | End LSN. |
| startlsn | text | Start LSN of the Xlog. |
| endlsn | text | End LSN of the Xlog. |
| prelsn | text | Start LSN of the previous Xlog. |
| xid | xid | Xlog transaction ID. |
| datalen | int4 | Xlog data length, in bytes. |
| totallen | int4 | Xlog length, in bytes. |
| type | text | Xlog type. |
| desc | text | Xlog content. |
| blkref | text | Relfilenode associated with the Xlog. |

 NOTE

- The length of the **xlogdescribe** field varies depending on the Xlog type. The **pg_xlogdump()** function truncates the field and retains only the first 64 bytes.
- You can find the target Xlog and use the **pg_xlog_display_one_lsn()** function to view the complete Xlog content.

Example:

```
SELECT * FROM pg_xlogdump('0/101EF70','0/101F590');
 node_name | startlsn | endlsn | prelsn | xlog_tid | datalen | totalen | xlogtype | xlogdescribe
-----+-----+-----+-----+-----+-----+-----+-----+-----
 datanode1 | 0/101EF70 | 0/101EFC0 | 0/101EF18 |          | 6 | 2 | 80 | Btree | insert leaf: off 1 |
 blkrel #0: rel 1663/1/2674, fork main, blk 1
 datanode1 | 0/101EFC0 | 0/101F018 | 0/101EF70 |          | 6 | 3 | 88 | Heap | insert: off 2 flags 0 |
 blkrel #0: rel 1663/1/2608, fork main, blk 0
 datanode1 | 0/101F018 | 0/101F068 | 0/101EFC0 |          | 6 | 2 | 80 | Btree | insert leaf: off 1 |
 blkrel #0: rel 1663/1/2673, fork main, blk 1
 datanode1 | 0/101F068 | 0/101F0B8 | 0/101F018 |          | 6 | 2 | 80 | Btree | insert leaf: off 2 |
 blkrel #0: rel 1663/1/2674, fork main, blk 1
 datanode1 | 0/101F0B8 | 0/101F110 | 0/101F068 |          | 6 | 3 | 88 | Heap | insert: off 3 flags 0 |
 blkrel #0: rel 1663/1/2608, fork main, blk 0
 datanode1 | 0/101F110 | 0/101F160 | 0/101F0B8 |          | 6 | 2 | 80 | Btree | insert leaf: off 1 |
 blkrel #0: rel 1663/1/2673, fork main, blk 1
 datanode1 | 0/101F160 | 0/101F1B0 | 0/101F110 |          | 6 | 2 | 80 | Btree | insert leaf: off 3 |
 blkrel #0: rel 1663/1/2674, fork main, blk 1
 datanode1 | 0/101F1B0 | 0/101F208 | 0/101F160 |          | 6 | 3 | 88 | Heap | insert: off 4 flags 0 |
 blkrel #0: rel 1663/1/2608, fork main, blk 0
 datanode1 | 0/101F208 | 0/101F258 | 0/101F1B0 |          | 6 | 2 | 80 | Btree | insert leaf: off 1 |
 blkrel #0: rel 1663/1/2673, fork main, blk 1
 datanode1 | 0/101F258 | 0/101F2A8 | 0/101F208 |          | 6 | 2 | 80 | Btree | insert leaf: off 4 |
 blkrel #0: rel 1663/1/2674, fork main, blk 1
 datanode1 | 0/101F2A8 | 0/101F300 | 0/101F258 |          | 6 | 3 | 88 | Heap | insert: off 5 flags 0 |
 blkrel #0: rel 1663/1/2608, fork main, blk 0
 datanode1 | 0/101F300 | 0/101F350 | 0/101F2A8 |          | 6 | 2 | 80 | Btree | insert leaf: off 1 |
 blkrel #0: rel 1663/1/2673, fork main, blk 1
 datanode1 | 0/101F350 | 0/101F3A0 | 0/101F300 |          | 6 | 2 | 80 | Btree | insert leaf: off 5 |
 blkrel #0: rel 1663/1/2674, fork main, blk 1
 datanode1 | 0/101F3A0 | 0/101F3F8 | 0/101F350 |          | 6 | 3 | 88 | Heap | insert: off 6 flags 0 |
 blkrel #0: rel 1663/1/2608, fork main, blk 0
 datanode1 | 0/101F3F8 | 0/101F448 | 0/101F3A0 |          | 6 | 2 | 80 | Btree | insert leaf: off 1 |
 blkrel #0: rel 1663/1/2673, fork main, blk 1
 datanode1 | 0/101F448 | 0/101F498 | 0/101F3F8 |          | 6 | 2 | 80 | Btree | insert leaf: off 5 |
 blkrel #0: rel 1663/1/2674, fork main, blk 1
 datanode1 | 0/101F498 | 0/101F4F0 | 0/101F448 |          | 6 | 3 | 88 | Heap | insert: off 7 flags 0 |
 blkrel #0: rel 1663/1/2608, fork main, blk 0
 datanode1 | 0/101F4F0 | 0/101F540 | 0/101F498 |          | 6 | 2 | 80 | Btree | insert leaf: off 1 |
 blkrel #0: rel 1663/1/2673, fork main, blk 1
 datanode1 | 0/101F540 | 0/101F590 | 0/101F4F0 |          | 6 | 2 | 80 | Btree | insert leaf: off 6 |
 blkrel #0: rel 1663/1/2674, fork main, blk 1
(19 rows)
```

21.7.32 Trigger Functions

pg_get_triggerdef(oid)

Description: Obtains the definition information of a trigger.

Parameter: OID of the trigger to be queried

Return type: text

Example:

```
SELECT pg_get_triggerdef(oid) FROM pg_trigger;
          pg_get_triggerdef
-----
CREATE TRIGGER insert_trigger BEFORE INSERT ON test_trigger_src_tbl FOR EACH ROW EXECUTE
PROCEDURE tri_insert_func()
(1 row)
```

pg_get_triggerdef(oid, boolean)

Description: Obtains the definition information of a trigger.

Parameter: OID of the trigger to be queried and whether it is displayed in pretty mode

Return type: text

NOTE

The Boolean parameters take effect only when the WHEN condition is specified during trigger creation.

Example:

```
SELECT pg_get_triggerdef(oid,true) FROM pg_trigger;
          pg_get_triggerdef
-----
CREATE TRIGGER insert_trigger BEFORE INSERT ON test_trigger_src_tbl FOR EACH ROW EXECUTE
PROCEDURE tri_insert_func()
(1 row)

SELECT pg_get_triggerdef(oid,false) FROM pg_trigger;
          pg_get_triggerdef
-----
CREATE TRIGGER insert_trigger BEFORE INSERT ON test_trigger_src_tbl FOR EACH ROW EXECUTE
PROCEDURE tri_insert_func()
(1 row)
```

21.7.33 XML Functions

21.7.33.1 Generating XML Content

Expressions of functions and class functions in this section can be used to generate XML content from SQL data. This method is used to format query results into XML documents for processing in client applications.

XMLPARSE ({ DOCUMENT | CONTENT } *value*)

Description: Generates an XML value from character data.

Return type: XML

Example:

```
SELECT xmlparse(document '<foo>bar</foo>');
      xmlparse
-----
<foo>bar</foo>
(1 row)
```


XMLSERIALIZE ({ DOCUMENT | CONTENT } *value AS type*

Description: Generates a string from XML values.

Return type: type, which can be character, character varying, or text (or its alias)

Example:

```
SELECT xmlserialize(content 'good' AS CHAR(10));
xmlserialize
-----
good
(1 row)
```

xmlcomment(text)

Description: Creates an XML note that uses the specified text as the content. The text cannot contain (--) or end with a hyphen (-). If the parameter is null, the result is also null.

Return type: XML

Example:

```
SELECT xmlcomment('hello');
xmlcomment
-----
<!--hello-->
(1 row)
```

xmlconcat(xml[, ...])

Description: Concatenates a list of XML values into a single value. Null values are ignored. If all parameters are null, the result is also null.

Return type: XML

Example:

```
SELECT xmlconcat('<abc/>', '<bar>foo</bar>');
xmlconcat
-----
<abc/><bar>foo</bar>
(1 row)
```

Note: If XML declarations exist and they are the same XML version, the result will use the version. Otherwise, the result does not use any version. If all XML values have the **standalone** attribute whose status is **yes**, the **standalone** attribute in the result is **yes**. If at least one XML value's **standalone** attribute is **no**, the **standalone** attribute in the result is **no**. Otherwise, the result does not contain the **standalone** attribute.

Example:

```
SELECT xmlconcat('<?xml version="1.1"?><foo/>', '<?xml version="1.1" standalone="no"?><bar/>');
xmlconcat
-----
<?xml version="1.1"?><foo/><bar/>
(1 row)
```

xmlelement(name name [, xmlattributes(value [AS attname] [, ...])] [, content, ...])

Description: Generates an XML element with the given name, attribute, and content.

Return type: XML

Example:

```
SELECT xmlelement(name foo);
xmlelement
-----
<foo/>
(1 row)

SELECT xmlelement(name foo, xmlattributes('xyz' as bar));
xmlelement
-----
<foo bar="xyz"/>
(1 row)

SELECT xmlelement(name foo, xmlattributes(current_date as bar), 'cont', 'ent');
xmlelement
-----
<foo bar="2023-08-16">content</foo>
(1 row)
```

Element and attribute names without valid XML names are escaped by replacing invalid characters with the sequence **_xHHHH_**, where **HHHH** is the Unicode code points of the character expressed in hexadecimal format. Example:

```
SELECT xmlelement(name "foo$bar", xmlattributes('xyz' as "a&b"));
xmlelement
-----
<foo_x0024_bar a_x0026_b="xyz"/>
```

If the attribute value is a column reference, you do not need to specify an explicit attribute name, and the column name is used as the attribute name by default. In other cases, the attribute must be given an explicit name. So this example is legal:

```
CREATE TABLE test (a xml, b xml);
SELECT xmlelement(name test, xmlattributes(a, b)) FROM test;
```

But these are illegal:

```
SELECT xmlelement(name test, xmlattributes('constant'), a, b) FROM test;
SELECT xmlelement(name test, xmlattributes(func(a, b))) FROM test;
```

The element content, if specified, will be formatted based on its data type. If the content itself is of the XML type, a complex XML document will be constructed.

Example:

```
SELECT xmlelement(name foo, xmlattributes('xyz' as bar), xmlelement(name
abc), xmlcomment('test'), xmlelement(name xyz));
xmlelement
-----
<foo bar="xyz"><abc/><!--test--><xyz/></foo>
```

Other types of content will be formatted into valid XML character data. This means that special characters **<**, **>**, and **&** will be converted to entities. Binary data (data type **bytea**) is represented as Base64 or hexadecimal code, depending on the configuration parameter **xmlbinary**.

xmlforest(content [AS name] [, ...])

Description: Generates an XML forest (sequence) of an element with a given name and content.

Return type: XML

Example:

```
SELECT xmlforest('abc' AS foo, 123 AS bar);
       xmlforest
-----
<foo>abc</foo><bar>123</bar>
(1 row)

SELECT xmlforest(table_name, column_name) FROM ALL_TAB_COLUMNS WHERE schema = 'pg_catalog';
       xmlforest
-----
<table_name>pg_authid</table_name><column_name>rolsuper</column_name>
<table_name>pg_authid</table_name><column_name>rolinherit</column_name>
<table_name>pg_authid</table_name><column_name>rolcreatorole</column_name>
```

xmlpi(name target [, content])

Description: Creates an XML processing instruction. The content cannot contain the character sequence of ?>.

Return type: XML

Example:

```
SELECT xmlpi(name php, 'echo "hello world";');
       xmlpi
-----
<?php echo "hello world";?>
(1 row)
```

xmlroot(xml, version text | no value [, standalone yes|no|no value])

Description: Modifies the attributes of the root node of an XML value. If a version is specified, it replaces the value in the version declaration of the root node. If a **standalone** value is specified, it replaces the **standalone** value in the root node.

Return type: XML

Example:

```
SELECT xmlroot(xmlparse(document '<?xml version="1.0" standalone="no"?><content>abc</content>'),
              version '1.1', standalone yes);
       xmlroot
-----
<?xml version="1.1" standalone="yes"?><content>abc</content>
(1 row)
```

xmlagg(xml)

Description: The **xmlagg** function is an aggregate function that concatenates input values.

Return type: XML

Example:

```
CREATE TABLE test (y int, x xml);
INSERT INTO test VALUES (1, '<foo>abc</foo>');
INSERT INTO test VALUES (2, '<bar/>');

SELECT xmlagg(x) FROM test;
      xmlagg
-----
<bar/><foo>abc</foo>
(1 row)
```

Add an **ORDER BY** clause to an aggregate call to determine the concatenation sequence. For example:

```
SELECT xmlagg(x ORDER BY y DESC) FROM test;
      xmlagg
-----
<bar/><foo>abc</foo>
(1 row)
```

21.7.33.2 XML Predicates

The functions in this section check the attributes of an XML value.

xml IS DOCUMENT

Description: IS DOCUMENT returns true if the XML value of the parameter is a correct XML document; if the XML document is incorrect, false is returned. If the parameter is null, a null value is returned.

Return type: bool

```
SELECT '<abc/>' is document;
?column?
-----
t
(1 row)
```

xml IS NOT DOCUMENT

Description: Returns **true** if the XML value of the parameter is not a correct XML document. If the XML document is correct, **false** is returned. If the parameter is null, a null value is returned.

Return type: bool

```
SELECT 'abc' is document;
?column?
-----
f
(1 row)
```

XMLEXISTS(text PASSING [BY REF] xml [BY REF])

Description: If the **xpath** expression in the first parameter returns any node, the **XMLEXISTS** function returns true. Otherwise, the function returns false. (If any parameter is null, the result is null.) The BY REF clause is invalid and is used to maintain SQL compatibility.

Return type: bool

Example:

```
SELECT xmlexists('//*[text() = "TScity"]' PASSING BY REF '<towns><town>TScity</town><town>TOcity</town></towns>');
xmlexists
-----
t
(1 row)
```

xml_is_well_formed(text)

Description: Checks whether a text string is a well-formatted XML value and returns a Boolean result. If the **xmloption** parameter is set to **DOCUMENT**, the document is checked. If the **xmloption** parameter is set to **CONTENT**, the content is checked.

Return type: bool

Example:

```
SET xmloption TO DOCUMENT;
SET

SELECT xml_is_well_formed('<>');
xml_is_well_formed
-----
f
(1 row)

SELECT xml_is_well_formed('<abc/>');
xml_is_well_formed
-----
t
(1 row)

SET xmloption TO CONTENT;
SELECT xml_is_well_formed('abc');
xml_is_well_formed
-----
t
(1 row)
```

xml_is_well_formed_document(text)

Description: Checks whether a text string is a well-formatted text and returns a Boolean result.

Return type: bool

Example:

```
SELECT xml_is_well_formed_document('<test:foo xmlns:test="http://test.com/test">bar</test:foo>');
xml_is_well_formed_document
-----
t
(1 row)

SELECT xml_is_well_formed_document('<test:foo xmlns:test="http://test.com/test">bar</my:foo>');
xml_is_well_formed_document
-----
f
(1 row)
```

xml_is_well_formed_content(text)

Description: Checks whether a text string is a well-formatted content and returns a Boolean result.

Return type: bool

Example:

```
SELECT xml_is_well_formed_content('content');
xml_is_well_formed_content
-----
t
(1 row)

SELECT xml_is_well_formed_content('<content');
xml_is_well_formed_content
-----
f
(1 row)
```

21.7.33.3 Processing XML

To process values of the XML data type, DWS provides the **xpath** and **xpath_exists** functions, as well as the **XMLTABLE** table function.

xpath(xpath, xml [, nsarray])

Description: Returns an array of XML values corresponding to the set of nodes produced by the **xpath** expression. If the **xpath** expression returns a scalar value instead of a set of nodes, an array of individual elements is returned.

The second parameter **xml** must be a complete XML document, which must have a root node element.

The third parameter is optional and is an array mapping of a namespace. The array should be a two-dimensional text array, and the length of the second dimension should be 2. (It should be an array of arrays, each containing exactly two elements). The first element of each array item is the alias of the namespace name, and the second element is the namespace URI. The alias provided in this array does not have to be the same as the alias used in the XML document itself. In other words, in the context of both XML documents and **xpath** functions, aliases are local.

Return type: XML value array

Example:

```
SELECT xpath('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>', ARRAY[ARRAY['my',
'http://example.com']]);
xpath
-----
{test}
(1 row)
```

Process the default (anonymous) namespace:

```
SELECT xpath('/mydefns:b/text()', '<a xmlns="http://example.com"><b>test</b></a>', ARRAY[ARRAY['mydefns', 'http://example.com']]);
xpath
-----
{test}
(1 row)
```

xpath_exists(xpath, xml [, nsarray])

Description: The **xpath_exists** function is a special form of the **xpath** function. This function does not return an XML value that satisfies the **xpath** function; it

returns a Boolean value indicating whether the query is satisfied. This function is equivalent to the standard **XMLEXISTS** predicate, but it also provides support for a namespace mapping parameter.

Return type: bool

Example:

```
SELECT xpath_exists('/my:a/text()', '<my:a xmlns:my="http://example.com">test</my:a>',  
ARRAY[ARRAY['my', 'http://example.com']]);  
xpath_exists  
-----  
t  
(1 row)
```

xmltable

Description: Generates a table based on the input XML data, **XPath** expression, and column definition. An **xmltable** is similar to a function in syntax, but it can appear only as a table in the FROM clause of a query.

Return value: setof record

Syntax:

```
XMLTABLE ( [ XMLNAMESPACES ( namespace_uri AS namespace_name [, ...] ), ]  
          row_expression PASSING [ BY { REF | VALUE } ]  
          document_expression [ BY { REF | VALUE } ]  
          COLUMNS name { type [ PATH column_expression ] [ DEFAULT default_expression ] [ NOT NULL |  
NULL ] | FOR ORDINALITY }  
          [, ...]  
          )
```

Parameter:

- The optional XMLNAMESPACES clause is a comma-separated list of namespace definitions, where each **namespace_uri** is a text-type expression and each **namespace_name** is a simple identifier. XMLNAMESPACES specifies the XML namespaces used in the document and their aliases. The default namespace declaration is not supported.
- The mandatory parameter **row_expression** is an **XPath** 1.0 expression. This expression calculates the sequence of XML nodes based on the provided XML document **document_expression**. The sequence is the sequence of converting **xmltable** to output lines. If the **document_expression** value is **NULL** or an empty node set generated by **row_expression**, no line is returned.
- The **document_expression** parameter is used to input an XML document. The input document must be in the XML format. XML segment data or XML documents in incorrect formats are not supported. The BY REF and BY VALUE clauses do not take effect. They are used only to implement SQL standard compatibility.
- The COLUMNS clause specifies the column list definition in the output table. The column name and column data type are mandatory, and the path, default value, and whether the clause is empty are optional.
 - **column_expression** of a column is an **XPath** 1.0 expression used to calculate the value of the column extracted from the current row based on **row_expression**. If **column_expression** is not specified, the field name is used as an implicit path.

- A column can be marked as **NOT NULL**. If **column_expression** in the **NOT NULL** column does not return any data, and there is no **DEFAULT** clause or the calculation result of **default_expression** is **NULL**, an error is reported.
- The columns marked as **FOR ORDINALITY** are filled with row numbers starting from 1. The sequence is the node sequence retrieved from the **row_expression** result set. A maximum of one column can be marked as **FOR ORDINALITY**.

NOTICE

XPath 1.0 does not specify the order for nodes, so the order in which results are returned depends on the order in which data is obtained.

Example:

```
SELECT * FROM XMLTABLE('/ROWS/ROW'  
PASSING '<ROWS><ROW id="1"><CITY_ID>1002a</CITY_ID><CITY_NAME>snowcity</CITY_NAME></  
ROW><ROW id="2"><CITY_ID>1003b</CITY_ID><CITY_NAME>icecity</CITY_NAME></ROW><ROW  
id="3"><CITY_ID>1004c</CITY_ID><CITY_NAME>windcity</CITY_NAME></ROW></ROWS>'  
COLUMNS id INT PATH '@id',  
ordinality FOR ORDINALITY,  
CITY_id TEXT PATH 'CITY_ID',CITY_name TEXT PATH 'CITY_NAME' NOT NULL);  
id | ordinality | city_id | city_name  
-----  
1 | 1 | 1002a | snowcity  
2 | 2 | 1003b | icecity  
3 | 3 | 1004c | windcity  
(3 rows)
```

21.7.33.4 Mapping a Table to XML

table_to_xml(tbl regclass, nulls boolean, tableforest boolean, targetns text)

Description: Maps the contents of a table to XML values.

Return type: XML

table_to_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)

Description: Maps a relational table schema to an XML schema document.

Return type: XML

table_to_xml_and_xmlschema(tbl regclass, nulls boolean, tableforest boolean, targetns text)

Description: Maps a relational table to XML values and schema documents.

Return type: XML

query_to_xml(query text, nulls boolean, tableforest boolean, targetns text)

Description: Maps the contents of an SQL query to XML values.

Return type: XML

query_to_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)

Description: Maps an SQL query into an XML schema document.

Return type: XML

query_to_xml_and_xmlschema(query text, nulls boolean, tableforest boolean, targetns text)

Description: Maps SQL queries to XML values and schema documents.

Return type: XML

cursor_to_xml(cursor refcursor, count int, nulls boolean, tableforest boolean, targetns text)

Description: Maps a cursor query to an XML value.

Return type: XML

cursor_to_xmlschema(cursor refcursor, nulls boolean, tableforest boolean, targetns text)

Description: Maps a cursor query to an XML schema document.

Return type: XML

schema_to_xml(schema name, nulls boolean, tableforest boolean, targetns text)

Description: Maps a table in a schema to an XML value.

Return type: XML

schema_to_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

Description: Maps a table in a schema to an XML schema document.

Return type: XML

schema_to_xml_and_xmlschema(schema name, nulls boolean, tableforest boolean, targetns text)

Description: Maps a table in a schema to an XML value and a schema document.

Return type: XML

database_to_xml(nulls boolean, tableforest boolean, targetns text)

Description: Maps a database table to an XML value.

Return type: XML

database_to_xmlschema(nulls boolean, tableforest boolean, targetns text)

Description: Maps a database table to an XML schema document.

Return type: XML

database_to_xml_and_xmlschema(nulls boolean, tableforest boolean, targetns text)

Description: Maps database tables to XML values and schema documents.

Return type: XML

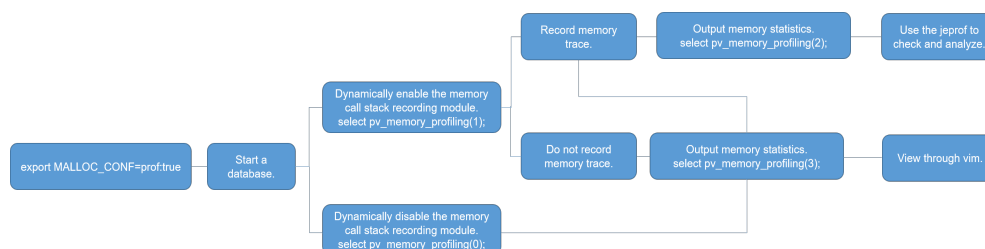
NOTE

The parameters for mapping a table to an XML value are described as follows:

- **tbl**: table name.
- **nulls**: indicates whether the output contains null values. If the value is **true**, the null value in the column is `<columnname xsi:nil="true"/>`. If the value is **false**, the columns containing null values are omitted from the output.
- **tableforest**: If this parameter is set to **true**, XML fragments are generated. If this parameter is set to **false**, XML files are generated.
- **targetns**: specifies the XML namespace of the desired result. If this parameter is not specified, an empty string is passed.
- **query**: SQL query statement
- **cursor**: cursor name
- **count**: amount of data obtained from the cursor
- **schema**: schema name

21.7.34 Call Stack Recording Functions

The **pv_memory_profiling(type int)** and environment variable **MALLOC_CONF** are used by DWS to control the enabling and disabling of the memory allocation call stack recording module and the output of the process-level memory call stack. The following figure illustrates the process.



MALLOC_CONF

The environment variable **MALLOC_CONF** is used to enable the monitoring module. It is in the `${BIGDATA_HOME}/mppdb/.mppdbgs_profile` file and is enabled by default. Note the following points:

- Restart the database after modifying this environment variable.

- If **om_monitor** is enabled in the cluster, restart the **om_monitor** process and then the database after setting this environment variable, so that the setting can take effect.
- This environment variable can be set on all servers in the cluster or on some servers where the module needs to be enabled. For the GaussDB process, each process determines whether to enable the module based on the **MALLOC_CONF** environment variable.

Commands for enabling and disabling **MALLOC_CONF**:

- Enabling the monitoring module:
`export MALLOC_CONF=prof:true`
- Disabling the monitoring module:
`export MALLOC_CONF=prof:false`

pv_memory_profiling (type int)

Parameter description: Controls the backtrace recording and output of memory allocation functions such as **malloc** in the gaussdb process.

Value range: a positive integer from 0 to 3.

Table 21-84 Values and descriptions of **pv_memory_profile**

| pv_memory_profiling Value | Description |
|----------------------------------|--|
| 0 | Disables the memory trace function and does not record information of call stacks such as malloc . |
| 1 | Enables the memory trace function to record information of call stacks such as malloc . |
| 2 | Outputs trace logs of call stacks such as malloc . <ul style="list-style-type: none">• Output path: /proc/pid/cwd directory. <i>pid</i> indicates the ID of the GaussDB process.• Output log name format: jeprof.<pid>.*.heap, where <i>pid</i> indicates the ID of the GaussDB process and * indicates the unique sequence number of the output trace log, for example, jeprof.195473.0.u0.heap. |
| 3 | Outputs memory statistics. <ul style="list-style-type: none">• Output path: /proc/pid/cwd directory. <i>pid</i> indicates the ID of the GaussDB process.• Log name format: <i>Node name + Process ID + Time + heap_stats + .out</i>. You can use vim to open the file. |

Return type: Boolean

Note:

- If the function is called successfully, **true** is returned. Otherwise, **false** is returned.
- If **Memory profiling failed, check if \$MALLOC_CONF contain 'prof:true'**. is displayed, it indicates that the module is used when **MALLOC_CONF=prof:true** is not set. In this case, you need to set the environment variable.
- If **Type %d is not supported. The valid range is 0-3.** is displayed, the parameter value is incorrect. The correct values are **0, 1, 2, and 3**.
- If "Memory profiling failed, input type is %d, failed number is %d." is displayed, contact technical support.

Outputting Memory Call Stack Information

Procedure:

- Step 1** Execute the following statement to output the memory call stack information and output the **trace** file in the directory where the GaussDB process is located:

```
SELECT * FROM pv_memory_profiling(2);
```

- Step 2** Use the jeprof tool provided by jemalloc to parse log information.

Method 1: Output in text format.

```
jeprof --text --show_bytes $GAUSSHOME/bin/gaussdb trace file 1 >prof.txt
```

Method 2: Export the report in PDF format.

```
jeprof --pdf --show_bytes $GAUSSHOME/bin/gaussdb trace file 1 > prof.pdf
```

NOTE

- To parse the memory call stack information, you need to use the GaussDB source code for analysis. You need to send the **trace** file to R&D engineers for analysis.
- To analyze the **trace** file, you need to use the jeprof tool, which is generated by jemalloc. The Perl environment is required for using the tool. To generate PDF calling diagrams, you need to install the GraphViz tool that matches the OS.

----End

Example

Log in as the system administrator, set environment variables, and start the database.

```
export MALLOC_CONF=prof:true
```

Disable the memory trace recording function when the database is running.

```
postgres=#SELECT pv_memory_profiling(0);
pv_memory_profiling
-----
t
(1 row)
```

Enable the memory trace recording function when the database is running.

```
postgres=#SELECT pv_memory_profiling(1);
pv_memory_profiling
-----
t
(1 row)
```

Output memory trace records.

```
postgres=#SELECT pv_memory_profiling(2);
pv_memory_profiling
-----
t
```

(1 row)

Generate the trace file in text or PDF format in the directory where the GaussDB process is located.

```
jeprof --text --show_bytes $GAUSSHOME/bin/gaussdb trace file 1 >prof.txt
jeprof --pdf --show_bytes $GAUSSHOME/bin/gaussdb trace file 1 > prof.pdf
```

Outputs memory statistics. Execute the following statement to generate the memory statistics file in the directory where the GaussDB process is located. The file can be directly read.

```
postgres=#SELECT pv_memory_profiling(3);
pv_memory_profiling
-----
t
(1 row)
```

21.7.35 Hudi System Functions

Hudi system functions are supported only by 9.1.0.100 and later cluster versions.

pg_show_custom_settings()

Description: Queries details about Hudi foreign table parameter settings.

Return type: SETOF record

Example:

```
SELECT * FROM pg_show_custom_settings();
      name      | setting | unit | category | short_desc |
extra_desc | context | vartype | source | min_val | max_val |
enumvals | boot_val | reset_val | sourcefile | sourceline
-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+
hoodie.public.hudi_mor_ft.consume.ending.timestamp | 20230404172329544 | | Customized Options |
GUC placeholder variable | | user | string | session | | |
| | | | | | | | | | | | |
hoodie.public.hudi_mor_ft.consume.mode | incremental | | Customized Options | GUC
placeholder variable | | user | string | session | | |
| | | | | | | | | | | | |
hoodie.public.hudi_mor_ft.consume.start.timestamp | 20230404172329543 | | Customized Options |
GUC placeholder variable | | user | string | session | | |
| | | | | | | | | | | | |
(3 rows)
```

hudi_get_options(regclass)

Description: Queries the properties of a Hudi foreign table (hoodie.properties). It is represented by a key-value pair.

Return type: SETOF record

Example:

```
select * from hudi_get_options('public.hudi_mor_ft');
      key                | value
-----+-----
hoodie.table.precombine.field | col_int
hoodie.datasource.write.drop.partition.columns | false
hoodie.table.partition.fields | 
hoodie.table.type | MERGE_ON_READ
hoodie.archive.log.folder | archived
hoodie.compaction.payload.class | 
org.apache.hudi.common.model.OverwriteWithLatestAvroPayload
hoodie.timeline.layout.version | 1
hoodie.table.version | 4
hoodie.table.recordkey.fields | col_bigint
hoodie.database.name | default
hoodie.datasource.write.partitionpath.urlencode | false
hoodie.table.name | lt_test_mor_014
hoodie.table.keygenerator.class | org.apache.hudi.keygen.ComplexKeyGenerator
hoodie.datasource.write.hive_style_partitioning | true
hoodie.table.create.schema | 
{"type": "record", "name": "lt_test_mor_014_record", "namespace": "hoodie.lt_test_mor_014", "fields": [
  {"name": "_hoodie_commit_time", "type": [
    "string", "null"]}, {"name": "_hoodie_commit_seqno", "type": ["string", "null"]},
  {"name": "_hoodie_record_key", "type": ["string", "null"]}, {"name": "_hoodie_partition_path", "type": [
    "string", "null"]}, {"name": "_hoodie_file_name", "type": ["string", "null"]}, {"name": "col_bigint", "type": ["long", "null"]},
  {"name": "col_int", "type": ["int", "null"]}, {"name": "col_text", "type": ["string", "null"]}, {"name": "col_text2", "type": ["string", "null"]}]}
hoodie.table.checksum | 515660817
(16 rows)
```

hudi_get_max_commit(regclass)

Description: This function retrieves the most recent commit timestamp and data write time for the current Hudi external table.

Return type: record

Example:

```
SELECT * FROM hudi_get_max_commit('public.hudi_mor_ft');
max_commit | write_time
-----+-----
20221207141822 | 2022-12-07 14:18:30+08
(1 row)
```

hudi_get_commit(regclass, cstring, int)

Description: This function retrieves the timestamp and write time of the commit data for the current Hudi foreign table, starting from the specified commit and up to the *M*th commit. If the *M*th commit does not exist, the latest commit and the corresponding data write time are returned.

Return type: record

Example:

```
SELECT * FROM hudi_get_commit('public.hudi_mor_ft', '20230329174744657', 3);
   end_commit |      write_time
-----+-----
20230329174808908 | 2023-08-31 15:43:08+08
(1 row)
```

hudi_sync_task_submit(regclass, regclass)

Description: Submits an automatic Hudi synchronization task. The first input parameter is the synchronization target table, and the second input parameter is the Hudi foreign table. If the task is submitted successfully, the task ID is returned.

Return type: text

NOTE

- The synchronization target table must contain a primary key, which must be the same as the value of **hudi recordkey**.
- If the hudi table contains the **precombine** field, the synchronization target table must also contain the corresponding field.
- If the synchronization target table contains only the primary key (no other fields except the primary key), the synchronization task cannot be submitted normally.
- The user must have the **INSERT** and **UPDATE** permissions on the target table and the **SELECT** permission on the Hudi foreign table. Otherwise, the synchronization task cannot be submitted.

Example:

```
SELECT hudi_sync_task_submit('public.hudi_sync_i','public.hudi_mor_ft');
   hudi_sync_task_submit
-----
6465efe2-3ea1-0b00-dde5-b57dfb30fffe
(1 row)
```

hudi_sync_task_submit(regclass, regclass, interval)

Description: The function is the same as that of **hudi_sync_task_submit(regclass, regclass)**. The difference is that, in this function, you can specify an input parameter of the interval type to specify the task scheduling period. The value ranges from 5 seconds to 24 hours. If the task is submitted successfully, the task ID is returned.

Return type: text

NOTE

- The synchronization target table must contain a primary key, which must be the same as the value of **hudi recordkey**.
- If the hudi table contains the **precombine** field, the synchronization target table must also contain the corresponding field.
- If the synchronization target table contains only the primary key (no other fields except the primary key), the synchronization task cannot be submitted normally.
- The user must have the **INSERT** and **UPDATE** permissions on the target table and the **SELECT** permission on the Hudi foreign table. Otherwise, the synchronization task cannot be submitted.

Example:

```
SELECT hudi_sync_task_submit('public.hudi_sync_i','public.hudi_mor_ft','1 hour');
   hudi_sync_task_submit
```

```
-----  
6465efe2-3ea1-0b00-dde5-b57dfb30ffe  
(1 row)
```

hudi_sync_task_submit(regclass, regclass, text, text)

Description: The function is the same as that of **hudi_sync_task_submit(regclass, regclass)**. The difference is that you can specify two additional text input parameters to indicate the fields that you expect to be synchronized. The fields are separated by commas (,). Quotation marks and escape characters can be parsed. The number and sequence of fields in the two text parameters must be the same, because the synchronization will be performed based on the mapping between the fields in the two text parameters. If the task is submitted successfully, the task ID is returned.

Return type: text

NOTE

- The synchronization target table must contain a primary key, which must be the same as the value of **hudi recordkey**.
- If the hudi table contains the **precombine** field, the synchronization target table must also contain the corresponding field.
- If the synchronization target table contains only the primary key (no other fields except the primary key), the synchronization task cannot be submitted normally.
- The user must have the **INSERT** and **UPDATE** permissions on the target table and the **SELECT** permission on the Hudi foreign table. Otherwise, the synchronization task cannot be submitted.

Example:

```
SELECT hudi_sync_task_submit('public.hudi_sync_i','public.hudi_mor_ft','_hoodie_commit_time, col_bigint,  
col_text', '_hoodie_commit_time, col_bigint, col_text');  
hudi_sync_task_submit  
-----  
646610bc-cdd1-0d00-d07d-b57e89a0ffe  
(1 row)
```

hudi_sync_task_submit(regclass, regclass, text, text, interval)

Description: The function is the same as that of **hudi_sync_task_submit(regclass, regclass, text, text)**. The difference is that you can specify an additional input parameter of the interval type to specify the task scheduling period. The value ranges from 5 seconds to 24 hours.

Return type: text

NOTE

- The synchronization target table must contain a primary key, which must be the same as the value of **hudi recordkey**.
- If the hudi table contains the **precombine** field, the synchronization target table must also contain the corresponding field.
- If the synchronization target table contains only the primary key (no other fields except the primary key), the synchronization task cannot be submitted normally.
- The user must have the **INSERT** and **UPDATE** permissions on the target table and the **SELECT** permission on the Hudi foreign table. Otherwise, the synchronization task cannot be submitted.

Example:

```
SELECT hudi_sync_task_submit('public.hudi_sync_i','public.hudi_mor_ft','_hoodie_commit_time, col_bigint, col_text', '_hoodie_commit_time, col_bigint, col_text', '10 minute 30second');
      hudi_sync_task_submit
-----
646610bc-cdd1-0d00-d07d-b57e89a0fffe
(1 row)
```

hudi_show_sync_state()

Description: Obtains the synchronization status of the Hudi automatic synchronization task.

Return type: SETOF record

Example:

```
SELECT * FROM hudi_show_sync_state();
 target_tbl | source_ftbl | payload_type | precombine_key | latest_commit
-----+-----+-----+-----+-----
public.hudi_sync_i | public.hudi_mor_ft | org.apache.hudi.common.model.OverwriteWithLatestAvroPayload | col_int | 20230511114021573
(1 row)
```

hudi_sync(regclass, regclass)

Description: Stored procedure, which is invoked by the Hudi automatic synchronization task. Tasks submitted using **pg_catalog.hudi_sync_task_submit(regclass, regclass)** will execute the stored procedure. Upon successful execution of the command, the number of synchronized rows and timestamp will be displayed.

Return type: text

Example:

```
CALL hudi_sync('public.hudi_sync_i', 'public.hudi_mor_ft');
NOTICE: execute full sync
CONTEXT: PL/pgSQL function hudi_sync(regclass,regclass) line 11 at RETURN
      hudi_sync
-----
sync 1 rows up to 20230511114021573.
(1 row)
```

hudi_sync_custom(regclass, regclass, text)

Description: A stored procedure, which is the entry for invoking the Hudi automatic synchronization task. Users can customize the mapping between the fields in the target table and those in the data source table. Tasks submitted using **pg_catalog.hudi_sync_task_submit(regclass, regclass, text, text)** will execute the stored procedure. In the preceding information, **text** is a JSON string, indicating the synchronization mapping between two table fields. Upon successful execution of the command, the number of synchronized rows and timestamp will be displayed.

Return type: text

Example:

```
CALL hudi_sync_custom('public.hudi_sync_i', 'public.hudi_mor_ft', '{"_hoodie_commit_time" :
"_hoodie_commit_time", "col_bigint" : "col_bigint", "col_text" : "col_text"}');
NOTICE: execute full sync
CONTEXT: PL/pgSQL function hudi_sync_custom(regclass,regclass,text) line 14 at RETURN
        hudi_sync_custom
-----
sync 1 rows up to 20230511114021573.
(1 row)
```

hudi_set_sync_commit(regclass, regclass, text)

Description: Sets the start timestamp of the first synchronization of the Hudi automatic synchronization task to prevent resynchronization. The first parameter is the synchronization target table, the second parameter is the Hudi foreign table, and the third parameter is the expected synchronization start point. This function must be used before a synchronization task is submitted.

Return type: text

Example:

```
select hudi_set_sync_commit('public.hudi_sync_i', 'public.hudi_mor_ft', '20230511114021573');
NOTICE: set sync commit successfully, the next synchronization will start from 20230511114021573
CONTEXT: referenced column: hudi_set_sync_commit
        hudi_set_sync_commit
-----
20230511114021573
(1 row)
```

NOTE

You must have the **INSERT** and **UPDATE** permissions on the target table and the **SELECT** permission on the Hudi foreign table. Otherwise, you cannot set the synchronization progress.

hudi_set_sync_commit(text, text)

Description: Sets the start timestamp of the next synchronization of a Hudi automatic synchronization task. You can use it to sync historical data again or to skip some data. The first parameter is the task ID, and the second parameter is the expected start time of the next synchronization. This function can be used only after a synchronization task is submitted. Before using this function, you need to pause the task.

Return type: text

Example:

```
select hudi_set_sync_commit('6524c8e3-aae9-0000-5a14-be8ec000ffe', '20230511114021573');
NOTICE: set sync commit successfully, the next synchronization will start from 20230511114021573
CONTEXT: referenced column: hudi_set_sync_commit
        hudi_set_sync_commit
-----
20230511114021573
(1 row)
```

NOTE

- Only users who have the permission on the target task can invoke this function.
- Before invoking this function, ensure that the target task is paused and has been successfully executed at least once.

pg_task_show(text)

Description: Queries the information about the current automatic scheduling task. For a Hudi synchronization task, the input parameter should be **SQLonHudi**.

Return type: SETOF record

Example:

```
SELECT * FROM pg_task_show('SQLonHudi');
      task_id      |
-----+-----
what               | category_id | userid | is_broken |
interval | time_cons | start_time | end_time | parameter | task_rank | next_start_time
| next_end_time | last_log | failure_times
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+
64d257e9-1e9b-0d00-3ce3-7e61b5e0ffe | call pg_catalog.hudi_sync_custom('public.hudi_read_target',
'public.hudi_read101', '{"_hoodie_commit_seqno": "_hoodie_commit_seqno", "id": "id", "ts": "ts",
"long_field": "ts"}'); | SQLonHudi | 10 | f | '00:00:10' | 2023-08-08 22:58:15.846903+08
| 0 | 5 | 2023-08-08 22:58:15.846903+08 | 2023-08-08 22:58:24.846903+08 |
(1 row)
```

NOTE

The **last_log** and **failure_times** fields are used to record the status of the last task.

- The value of **last_log** is updated when the task is complete. If the task is successful, the content is cleared. If the task fails, the task failure log is recorded.
- The value of **failure_times** is updated at the end of each time window. It is set to **0** if the task is successful, incremented by 1 if the task fails, and unchanged if the task is not started in that round. This value helps identify when the first failure happens.

pg_task_remove(text)

Description: Deletes an automatic scheduling task. The input parameter is a task ID. The function returns the number of deleted tasks.

Return type: integer

Example:

```
SELECT pg_task_remove('64661705-8ada-0100-d07f-b57e89a0ffe');
pg_task_remove
-----
1
(1 row)
```

pg_task_pause(text)

Description: Suspend an automatic scheduling task. The input parameter is a task ID. The function returns the number of suspended tasks.

Return type: integer

Example:

```
SELECT pg_task_pause('64661705-8ada-0100-d07f-b57e89a0ffe');
pg_task_pause
-----
1
(1 row)
```

pg_task_resume(text)

Description: Resumes an automatic scheduling task. The input parameter is the ID of a suspended task. The function returns the number of resumed tasks.

Return type: integer

Example:

```
SELECT pg_task_resume('64661705-8ada-0100-d07f-b57e89a0fffe');
pg_task_resume
-----
1
(1 row)
```

pg_task_reset_interval(text, interval)

Description: Modifies the scheduling period of a synchronization task. The first input parameter is **task_id**, and the second input parameter the scheduling period, of which the value ranges from 5 seconds to 24 hours. The function returns the number of tasks whose periods are modified.

Return type: integer

Example:

```
select pg_task_reset_interval('64bfd69c-a016-0000-120e-1e802978fffe', '10 hours 30 minutes');
pg_task_reset_interval
-----
1
(1 row)
```

21.7.36 Funnel and Retention Functions

Funnel and retention functions are supported only in cluster 8.3.0 and later versions.

Context

Both funnel functions and retention functions are widely used tools for analyzing user behavior in product and data analysis. They are particularly useful for product managers, data scientists, and marketing personnel. These functions assist in analyzing the customer journey, comprehending user churn and conversion problems, and assessing the product's ability to consistently attract and retain customers.

- **Funnel analysis:** examines each step a customer takes from initial contact to final purchase. It helps product teams and marketers understand how users interact with products or services, as well as the conversion and churn rates at each step. The funnel analysis involves designating stages of the customer's journey from first hearing about a product to making a purchase. The whole process may include visiting a website, registering an account, and purchasing a product.
- **Retention analysis:** focuses on the percentage of users who continue using a product after a specified period of time. It calculates the number of users who remain active after a certain time point, such as user registration, first order placement, or initial use. Common retention analysis functions include daily,

weekly, and monthly retention rates, which help analyze whether users continue using the product after initially hearing about it.

WINDOW_FUNNEL

The **WINDOW_FUNNEL** function searches for an event chain in the sliding time window and counts the maximum number of consecutive events in the event chain.

Given a list of user-defined events, DWS finds the longest sequential match starting from the first event and returns the length of the match. Once the matching fails, the entire matching ends. Example:

Assume that the window is large enough:

- The condition events are c1, c2, and c3, but the user data is c1, c2, c3, and c4. So, c1, c2, and c3 are matched, and the return value of the function is 3.
- The condition events are c1, c2, and c3, but the user data is c4, c3, c2, and c1. So, c1 is matched, and the return value of the function is 1.
- The condition events are c1, c2, and c3, but the user data is c4 and c3. So, no event is matched, and the return value of the function is 0.

Function syntax

```
windowFunnel(window, mode, timestamp, cond1, cond2, ..., condN)
```

Input parameters

- **window**: bigint type. It indicates the size of the sliding time window. The unit is second.
- **mode**: text type. Currently, only the **Default** mode is supported. For other modes, an error is reported. In the **Default** mode, it matches as many events as possible, starting from the first event in the window.
- **timestamp**: time range when an event occurs. The **timestamp without time zone**, **timestamp with time zone**, **date**, **int** and **bigint** types are supported.
- **cond**: variable-length Boolean array. It indicates the condition. DWS supports only 1 to 32 conditions. If the number of conditions is not within the range, an error is reported.

Return values

level: int type. It indicates the maximum length of the event list that matches the condition.

Retention Functions

The retention function evaluates if an event meets each condition, starting from the first one.

Function syntax

```
retention(cond1, cond2, ..., cond32);
```

Input parameters

cond: variable-length Boolean array with a maximum length of 32 characters, indicating whether an event meets specific conditions. DWS supports only 1 to 32

conditions. If the number of conditions is not within the range, an error is reported.

Return values

retention condition: tinyint array type. It indicates the expression of the returned result, which is the same as the length of the input parameter **cond**. If the **cond1** and **condi** conditions are met, the *ith* value of the returned result is **1**. Otherwise, the *ith* value is **0**.

Other Retention-Related Functions

DWS supports functions **range_retention_count** and **range_retention_sum** as supplements to the retention function for better customer retention analysis.

- **range_retention_count**

range_retention_count calculates the retention rate of each user. This function returns an array, which can be used as the input parameter of the **range_retention_sum** function.

Function syntax

```
range_retention_count(is_first, is_active, dt, retention_interval, retention_granularity, output_format)
```

Input parameters

- **is_first**: indicates whether it is an initial behavior. The value is of the Boolean type. The value **true** indicates that it is an initial behavior, and the value **false** indicates that it is not an initial behavior.
- **is_active**: indicates whether it is a retention behavior. The value is of the Boolean type. The value **true** indicates that it is a retention behavior, and the value **false** indicates that it is not a retention behavior.
- **dt**: indicates the date when the behavior happens. The value is of the date type.
- **retention_interval**: indicates the storage interval. The value is of the array type. A maximum of 15 storage intervals are supported. For example, ARRAY[1,3,5,7,15,30].
- **retention_granularity**: indicates the retention granularity, which can be day, week, or month. The value is of the text type.
- **output_format**: indicates the output format. The value is of the text type and can be **normal** (default) or **expand** (daily retention details can be obtained).

Return value: BIGINT array of user retention information.

- **range_retention_sum**

range_retention_sum summarizes and calculates the daily (weekly/monthly) retention rate of all users.

Function syntax

```
range_retention_sum(range_retention_count_result)
```

Input parameter: return value of the **range_retention_count** function.

Return value: text array of user retention statistics.

Examples

Create the **funnel_test** table.

```
CREATE TABLE IF NOT EXISTS funnel_test
(
    user_id INT,
    event_type TEXT,
    event_time TIMESTAMP,
    event_timez TIMESTAMP WITH TIME ZONE,
    event_time_int BIGINT
);
```

Insert data.

```
INSERT INTO funnel_test VALUES
(1,'Browse','2021-01-31 11:00:00', '2021-01-31 11:00:00+08', 10),
(1,'Click','2021-01-31 11:10:00', '2021-01-31 11:10:00+07', 20),
(1,'Order','2021-01-31 11:20:00', '2021-01-31 11:20:00+06', 30),
(1,'Pay','2021-01-31 11:30:00', '2021-01-31 11:30:00+05', 40),
(2,'Order','2021-01-31 11:00:00', '2021-01-31 11:00:00+08', 11),
(2,'Pay','2021-01-31 11:10:00', '2021-01-31 11:10:00+08', 12),
(1,'Browse','2021-01-31 11:00:00', '2021-01-31 11:00:00+01', 50),
(3,'Browse','2021-01-31 11:20:00', '2021-01-31 11:20:00-04', 30),
(3,'Click','2021-01-31 12:00:00', '2021-01-31 12:00:00-04', 80),
(4,'Browse','2021-01-31 11:50:00', '2021-01-31 11:50:00-01', 1000),
(4,'Pay','2021-01-31 12:00:00', '2021-01-31 12:00:00-02', 900),
(4,'Order','2021-01-31 12:00:00', '2021-01-31 12:00:00-03', 1001),
(4,'Click','2021-01-31 12:00:00', '2021-01-31 12:00:00-04', 1001),
(5,'Browse','2021-01-31 11:50:00', '2021-01-31 11:50:00+08', NULL),
(5,'Click','2021-01-31 12:00:00', '2021-01-31 12:00:00+08', 776),
(5,'Order','2021-01-31 11:10:00', '2021-01-31 11:10:00+08', 999),
(6,'Browse','2021-01-31 11:50:00', '2021-01-31 11:50:00+01', -1),
(6,'Click','2021-01-31 12:00:00', '2021-01-31 12:00:00+02', -2),
(6,'Order','2021-01-31 12:10:00', '2021-01-31 12:00:00+03', -3);
```

Calculate the funnel for each user. In the **level** column of the following command output, **0** means that there were no events in the window period and **1** means that there was one event in the window period.

```
SELECT
    user_id,
    windowFunnel(
        0, 'default', event_timez,
        event_type = 'Browse', event_type = 'Click', event_type = 'Order', event_type = 'Pay'
    ) AS level
FROM funnel_test
GROUP BY user_id
ORDER BY user_id;
```

| user_id | level |
|---------|-------|
| 1 | 1 |
| 2 | 0 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |

(6 rows)

Calculate the funnel of each user and specify the length of the sliding time window as **NULL**. An error is reported.

```
SELECT
    user_id,
    windowFunnel(
        NULL, 'default', event_time,
        event_type = 'Browse', event_type = 'Click', event_type = 'Order', event_type = 'Pay'
    ) AS level
FROM funnel_test
GROUP BY user_id
ORDER BY user_id;
ERROR: Invalid parameter : window length or mode is null.
```

Calculate the funnel of each user and specify multiple conditions.

```
SELECT
  user_id,
  windowFunnel(
    40, 'default', date(event_time),
    true, true, false, true
  ) AS level
FROM funnel_test
GROUP BY user_id
ORDER BY user_id;
user_id | level
-----+-----
1 | 2
2 | 2
3 | 2
4 | 2
5 | 2
6 | 2
(6 rows)
```

Check the retention rate of each user in each event.

```
SELECT
  user_id,
  retention(
    event_type = 'Browse', event_type = 'Click', event_type = 'Order', event_type = 'Pay'
  ) AS r
FROM funnel_test
GROUP BY user_id
ORDER BY user_id ASC;
user_id | r
-----+-----
1 | {1,1,1,1}
2 | {0,0,0,0}
3 | {1,1,0,0}
4 | {1,1,1,1}
5 | {1,1,1,0}
6 | {1,1,1,0}
(6 rows)
```

Analyze the retention rate of each user in each event and set the first time point to **false**.

```
SELECT
  user_id,
  retention(
    false, event_type = 'Browse', event_type = 'Click', event_type = 'Order', event_type = 'Pay'
  ) AS r
FROM funnel_test
GROUP BY user_id
ORDER BY user_id ASC;
user_id | r
-----+-----
1 | {0,0,0,0,0}
2 | {0,0,0,0,0}
3 | {0,0,0,0,0}
4 | {0,0,0,0,0}
5 | {0,0,0,0,0}
6 | {0,0,0,0,0}
(6 rows)
```

Analyze the retention rate of all users in each event.

```
SELECT sum(r[1]), sum(r[2]), sum(r[3]), sum(r[4])
FROM
(
  SELECT
    retention(event_type = 'Browse', event_type = 'Click', event_type = 'Order', event_type = 'Pay') AS r
  FROM funnel_test
```



```
GROUP BY user_id
);
sum | sum | sum | sum
-----+-----+-----+-----
5 | 5 | 4 | 2
(1 row)
```

Create the **retention_test** table.

```
CREATE TABLE retention_test(
uid INT,
event TEXT,
event_time TIMESTAMP
);
```

Insert data.

```
INSERT INTO retention_test VALUES
(1, 'pay', '2024-05-01'),
(1, 'login', '2024-05-01'),
(1, 'pay', '2024-05-02'),
(1, 'login', '2024-05-02'),
(2, 'login', '2024-05-01'),
(3, 'login', '2024-05-02'),
(3, 'pay', '2024-05-03'),
(3, 'pay', '2024-05-04');
```

Gather statistics on the retention rate of each user in the payment event after one or two days.

```
WITH retention_count_info AS (
SELECT
uid,
range_retention_count(event = 'login', event = 'pay',
DATE(event_time), array[1, 2], 'day') AS info
FROM
retention_test
GROUP BY
uid
), retention_sum AS (
SELECT regexp_split_to_array(unnest(range_retention_sum(info)), ',') AS s
FROM retention_count_info
) SELECT to_date(s[1]::int) AS login_date,
s[3]::numeric / s[2]::numeric AS retention_rate1,
s[4]::numeric / s[2]::numeric AS retention_rate2
FROM retention_sum
ORDER BY login_date;
login_date | retention_rate1 | retention_rate2
-----+-----+-----
2024-05-01 00:00:00 | .50000000000000000000 | 0.00000000000000000000
2024-05-02 00:00:00 | .50000000000000000000 | .50000000000000000000
(2 rows)
```

21.7.37 EXTERNAL SCHEMA System Functions

System functions related to EXTERNAL SCHEMA are supported only by clusters of version 8.3.0 or later.

pg_get_external_schema_table_options(text, text)

Description: Obtains the options of an external schema table.

Input parameter: The first input parameter is the external schema name, and the second input parameter is the table name.

Return type: SETOF record

Example:

```
SELECT * FROM pg_get_external_schema_table_options('ex_lf', 'test_lf');
option_name | option_value
-----+-----
encoding    | utf8
format       | parquet
foldername   | /**/ /**/ /**/
(3 rows)
```

pg_get_external_schema_table_col(text, text)

Description: Obtains the column information of an external schema table.

Input parameter: The first input parameter is the external schema name, and the second input parameter is the table name.

Return type: SETOF record

Example:

```
SELECT * FROM pg_get_external_schema_table_col('ex_lf', 'test_lf');
col_name | col_type | part_col
-----+-----+-----
field_smallint | smallint | f
field_int | int | f
field_integer | int | f
field_bigint | bigint | f
field_float | float | f
field_double | double | f
field_double_precision | double | f
field_decimal | decimal(10,0) | f
field_numeric | decimal(10,0) | f
field_timestamp | timestamp | f
field_date | date | f
field_varchar | varchar(5) | f
field_char | char(5) | f
field_boolean | boolean | f
field_string | string | f
(15 rows)
```

21.7.38 Storage-Compute Decoupling Functions

21.7.38.1 Data Sharing Functions

pgxc_group_add_subscription(src_vw_name, target_vw_name)

Description: In a storage-compute decoupling architecture, a KV subscription relationship is established between logical clusters (Virtual Warehouses, VWs). Once established, the KV cache of the consumer VW periodically synchronizes the SST file incrementally from the OBS cudesc snapshot directory of the producer to the local host. This allows tables in the producer VW to be queried across VWs in the consumer VM. The created subscription relationship can be found in the **pgxc_group_subscription** table. This function is supported only by clusters of version 9.0.3 or later.

Return type: void

The following information is displayed:

Table 21-85 Columns returned by pgxc_group_add_subscription(src_vw_name, target_vw_name)

| Column | Type | Description |
|----------------|------|---|
| src_vw_name | text | Producer VW name, which is usually used as the VW to which data is written. |
| target_vw_name | text | Consumer VW name, which is usually used as the VW for reading data. |

Example:

```
SELECT pgxc_group_add_subscription('write_group', 'single_read_group');
pgxc_group_add_subscription
-----
(1 row)
```

pgxc_group_drop_subscription(src_vw_name, target_vw_name)

Description: To delete an established KV subscription relationship in a storage and compute separation architecture, remove the records from the **pgxc_group_subscription** table. After deletion, the consumer VW can no longer query tables in the producer VW across VWs. This function is supported only by clusters of version 9.0.3 or later.

Return type: void

The following information is displayed:

Table 21-86 Columns returned by pgxc_group_drop_subscription(src_vw_name, target_vw_name)

| Column | Type | Description |
|----------------|------|---|
| src_vw_name | text | Producer VW name, which is usually used as the VW to which data is written. |
| target_vw_name | text | Consumer VW name, which is usually used as the VW for reading data. |

Example:

```
SELECT pgxc_group_drop_subscription('write_group', 'single_read_group');
pgxc_group_drop_subscription
-----
(1 row)
```

21.7.38.2 Function for Obtaining File Information of Tablespaces in the Trash Directory

pgxc_tablespace_trash_info()

Description: Obtains the file statistics of the tablespaces whose filesystem is OBS in the trash directory in a storage-compute decoupling architecture. The information includes the number of files, total file size, and whether the automatic cleanup lifecycle policy is enabled for the trash directory. This function is available only for clusters of version 9.1.0.222 or later.

Return type: record

Return columns:

| Column | Type | Description |
|------------|------|--|
| spcname | text | Tablespace name. |
| file_count | int | Number of tablespace-related files in the trash directory. |
| total_size | int | Total size of tablespace files in the trash directory. |
| lifecycle | text | Whether the expiration policy for lifecycle is added to the trash directory. This field does not take effect in the current version. <ul style="list-style-type: none">● active: The policy is added.● inactive: The policy is not added. |

Example:

```
select * from pgxc_tablespace_trash_info();
spcname | file_count | total_size | lifecycle
-----+-----+-----+-----
cu_obs_tbs|      0    |      0    | inactive
(1 row)
```

21.7.39 Time Series Functions

[Table 21-87](#) lists the time series calculation functions supported by DWS.

Table 21-87 Time series calculation functions

| Time Series Calculation Function | Description |
|--------------------------------------|--|
| delta(field numeric) | Calculates the difference between two rows sorted by time. |

| Time Series Calculation Function | Description |
|--|--|
| spread(field numeric) | Calculates the difference between the maximum value and the minimum value in a specified period. |
| mode() within group (order by value anyelement) | Returns the value with the highest occurrence frequency for a given column. If multiple values have the same frequency, this function returns the smallest value among these values. |
| value_of_percentile(column float, percentile float, compression float) | Calculates the percentile. This is the approximation algorithm of percentile_cont . |
| percentile_of_value(column float, percentilevalue float, compression float) | Calculates a value based on a given percentile. This is the inverse function of value_of_percentile . |
| first(column1, column2) | Compares the values of column2 in a group, finds the minimum value, and outputs the value of column1 . |
| last(column1, column2) | Compares the values of column2 in a group, finds the maximum value, and outputs the corresponding value of column1 . |

delta(field numeric)

Description: Calculates the difference between two adjacent rows sorted by time in time series scenarios. It is used to monitor metrics such as traffic and speed.

Parameter: **field** indicates the column to be calculated.

CAUTION

The **delta** window function needs to be used together with the **over** window function. The **rows** statement in the **over** statement does not change the result of the **delta** function. For example, the results returned by **delta(value) over(order by time rows 1 preceding)** and **delta(value) over(order by time rows 3 preceding)** are the same.

Example:

```
SELECT
  delta(value) over (rows 1 preceding)
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz, 1),
    ('2019-07-12 00:01:00'::timestampz, 3),
    ('2019-07-12 00:02:00'::timestampz, 7))
  v(time,value) order by 1;
delta
-----
0
```

```
2
4
(3 rows)
```

spread(field numeric)

Description: Calculates the increment of each metric in a group. Generally, the increment is calculated after the metric is sorted by time.

Parameter: **field** indicates the column to be calculated.

CAUTION

If there are less than two tuples in each group, the returned result is **0**. Do not use this function with the **over** window function.

Example:

```
SELECT
  time, SPREAD(value)
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz, 1), ('2019-07-12 00:00:00'::timestampz, 3),
    ('2019-07-12 00:01:00'::timestampz, 3), ('2019-07-12 00:01:00'::timestampz, 6),
    ('2019-07-12 00:02:00'::timestampz, 7), ('2019-07-12 00:02:00'::timestampz, 9))
  v(time,value) group by time order by time;
  time      | spread
-----+-----
2019-07-12 00:00:00+08 |    2
2019-07-12 00:01:00+08 |    3
2019-07-12 00:02:00+08 |    2
(3 rows)
```

mode() within group (order by value anyelement)

Description: Returns the value with the highest occurrence frequency for a given column. If multiple values have the same frequency, this function returns the smallest value among these values.

Parameter: **value** indicates the column to be queried.

CAUTION

- This function must be used together with the **WITHIN GROUP** statement, or an error will be reported. The parameter of this function is placed after **ORDER BY** of the group.
- This function cannot be used together with the **over** clause.

Example:

```
SELECT
  mode() within group (order by value)
FROM
  (VALUES ('2019-07-12 00:00:00'::timestampz, 1),
    ('2019-07-12 00:01:00'::timestampz, 1),
    ('2019-07-12 00:02:00'::timestampz, 7))
  v(time,value);
mode
-----
```

```
1  
(1 row)
```

value_of_percentile(column float, percentile float, compression float)

Description: Returns an approximate percentile value for a given column in ascending order. It is an approximate result of **percentile_cont**, but the performance is better than that of **percentile_cont**.

Table 21-88 Parameters

| Parameter | Type | Description |
|-------------|-------|---|
| column | float | Column whose percentile is to be calculated |
| percentile | float | Percentile value. Value range: 0 to 1 |
| compression | float | Compression coefficient. The value ranges from 0 to 500 . The default value is 300 . A larger value indicates that more memory is occupied during function calculation and the result precision is higher. If the specified value is not within the value range, the value is regarded as 300. |

Example:

```
SELECT value_of_percentile(values, 0.8, 0) from TABLE;
```

percentile_of_value(column float, percentilevalue float, compression float)

Description: Returns the percentile of a given column in ascending order. This function is the inverse function of **value_of_percentile**.

Table 21-89 Parameters

| Parameter | Type | Description |
|-----------------|-------|---|
| column | float | Column whose percentile is to be calculated |
| percentilevalue | float | Value whose percentile is to be calculated |
| compression | float | Compression coefficient. The value ranges from 0 to 500 . The default value is 300 . A larger value indicates that more memory is occupied during function calculation and the result precision is higher. If the specified value is not within the value range, the value is regarded as 300. |

Example:

```
SELECT percentile_of_value(values, 80, 0) from TABLE;
```

first(column1, column2)

Description: Compares the values of **column2** in a group, finds the minimum value, and outputs the value of **column1**. It is an aggregation function.

Table 21-90 Parameters

| Parameter | Type | Description |
|-----------|-------------------------------|-------------------|
| column1 | bigint/text/double/numeric | Output column |
| column2 | timestamp/timestamptz/numeric | Comparison column |

Example: For details about the sample data, see the table definition and data in the [time_fill\(\)](#) expression.

Obtain the first idle value in each group sorted by time based on **scope_name**.

```
select first(idle, time_string) from dcs_cpu group by scope_name;
first
-----
  1
  3
(2 rows)
```

last(column1, column2)

Description: Compares the values of **column2** in a group, finds the maximum value, and outputs the value of **column1**. It is an aggregation function.

Table 21-91 Parameters

| Parameter | Type | Description |
|-----------|-------------------------------|-------------------|
| column1 | bigint/text/double/numeric | Output column |
| column2 | timestamp/timestamptz/numeric | Comparison column |

Example: For details about the sample data, see the table definition and data in the [time_fill\(\)](#) expression.

Obtain the last idle value in each group sorted by time based on **scope_name**.

```
select last(idle, time_string) from dcs_cpu group by scope_name;
last
-----
  3
  2
(2 rows)
```


Time Series Filling Functions

Table 21-92 lists the time series filling functions supported by DWS.

Table 21-92 Time series filling functions

| Time Series Filling Expression | Description |
|---|--|
| <code>time_fill(interval, time_column, start_time, end_time),
fill_last(agg_function(agg_column))</code> | Adds missing time data (aggregation result) after sorting by time column.
Uses the former value sorted by time to fill the latter value. |
| <code>time_fill(interval, time_column, start_time, end_time),
fill_first(agg_function(agg_column))</code> | Adds missing time data (aggregation result) after sorting by time column.
Uses the latter value sorted by time to fill the former value. |
| <code>time_fill(interval, time_column, start_time, end_time),
fill_avg(agg_function(agg_column))</code> | Adds missing time data (aggregation result) after sorting by time column.
Uses the former and latter values sorted by time to fill the current value. |



CAUTION

- The **time_fill** function needs to be used as an aggregation function. The **GROUP BY** clause needs to reference its own calculation result and cannot be nested with itself. The clause cannot be called multiple times in a single query, used as a lower-layer computing node, or used with the **WITHIN GROUP** clause.
- **time_fill** must be used together with **fill_avg**, **fill_first**, **fill_last**, or the **Agg** function.
- **time_fill** can only be used in **GROUP BY**. A **GROUP BY** clause that contains **time_fill** cannot contain other columns.
- **time_fill** cannot be used in other positions following **SELECT**, for example, after **WHERE** or other join conditions.
- The **start** timestamp must be smaller than the **finish** timestamp, and their interval must be greater than the value of **window_width**.
- All parameters cannot be null. Values of **start** and **finish** must be specified.

Table 21-93 Parameters

| Parameter | Type | Description |
|-----------|---|--------------------------|
| interval | INTERVAL. The minimum interval is 1 second. | Interval grouped by time |

| Parameter | Type | Description |
|--------------------------|---|---|
| time_column | TIMESTAMP or TIMESTAMPTZ | Interval grouped by a specified column |
| start_time | TIMESTAMP or TIMESTAMPTZ | Start time of a group |
| end_time | TIMESTAMP or TIMESTAMPTZ | End time of a group |
| agg_function(agg_column) | Specifies the agg function to aggregate specified columns, for example, max(col) . | Fills the agg result according to the specified filling method. |

time_fill(interval, time_column, start_time, end_time), fill_last(agg_function(agg_column))

Description: Uses the former value sorted by time to fill in the latter value.

Create the sample table **dc_s_cpu** and insert data into the table.

```
create table dcs_cpu(  
idle real,  
vcpu_num int,  
node text,  
scope_name text,  
server_ip text,  
iowait numeric,  
time_string timestamp  
)with (orientation=column) distribute by hash(node);  
insert into dcs_cpu VALUES(1.0,1,'node_a','scope_a','1.1.1.1',1.0,'2019-07-12 00:10:10');  
insert into dcs_cpu VALUES(2.0,2,'node_b','scope_a','1.1.1.2',2.0,'2019-07-12 00:12:10');  
insert into dcs_cpu VALUES(3.0,3,'node_c','scope_b','1.1.1.3',3.0,'2019-07-12 00:13:10');
```

Calculate the average value in a group at the interval of 1 min. Use the value of the former time segment to fill in the value of the latter time segment.

```
select time_fill(interval '1 min',time_string,'2019-07-12 00:09:00','2019-07-12 00:14:00'), fill_last(avg(idle))  
from dcs_cpu group by time_fill order by time_fill;  
   time_fill      | fill_last  
-----+-----  
2019-07-12 00:09:00 |  
2019-07-12 00:10:00 |      1  
2019-07-12 00:11:00 |      1  
2019-07-12 00:12:00 |      2  
2019-07-12 00:13:00 |      3  
2019-07-12 00:14:00 |      3  
(6 rows)
```

time_fill(interval, time_column, start_time, end_time), fill_first(agg_function(agg_column))

Description: Uses the latter value sorted by time to fill in the former value.

Example:

Calculate the average value in a group at the interval of 1 min. Use the value of the latter time segment to fill in the value of former time segment. (For details

about the sample data, see the table definition and data in [time_fill\(interval, time_column, start_time, end_time\), fill_last\(agg_function\(agg_column\)\).](#))

```
select time_fill(interval '1 min',time_string,'2019-07-12 00:09:00','2019-07-12 00:14:00'), fill_first(avg(idle))
from dcs_cpu group by time_fill order by time_fill;
   time_fill      | fill_first
-----+-----
2019-07-12 00:09:00 |      1
2019-07-12 00:10:00 |      1
2019-07-12 00:11:00 |      2
2019-07-12 00:12:00 |      2
2019-07-12 00:13:00 |      3
2019-07-12 00:14:00 |
(6 rows)
```

time_fill(interval, time_column, start_time, end_time), fill_avg(agg_function(agg_column))

Description: Uses the former and latter values sorted by time to fill in the current value.

Example:

Calculate the average value in a group at the interval of 1 min. Use the weighted average value of the former and latter values to fill in the current value. (For details about the sample data, see the table definition and data in [time_fill\(interval, time_column, start_time, end_time\), fill_last\(agg_function\(agg_column\)\).](#))

```
select time_fill(interval '1 min',time_string,'2019-07-12 00:09:00','2019-07-12 00:14:00'), fill_avg(avg(idle))
from dcs_cpu group by time_fill order by time_fill;
   time_fill      | fill_avg
-----+-----
2019-07-12 00:09:00 |      1
2019-07-12 00:10:00 |      1
2019-07-12 00:11:00 |     1.5
2019-07-12 00:12:00 |      2
2019-07-12 00:13:00 |      3
2019-07-12 00:14:00 |      3
(6 rows)
```

21.8 Expressions

21.8.1 Simple Expressions

Logical Expressions

[Logical Operators](#) lists the operators and calculation rules of logical expressions.

Comparative Expressions

[Comparison Operators](#) lists the common comparative operators.

In addition to comparative operators, you can also use the following sentence structure:

- BETWEEN operator
a BETWEEN x AND y is equivalent to **a >= x AND a <= y**.

a NOT BETWEEN x AND y is equivalent to **a < x OR a > y**.

- To check whether a value is null, use:
expression IS NULL
expression IS NOT NULL
or an equivalent (non-standard) sentence structure:
expression ISNULL
expression NOTNULL

NOTICE

Do not write **expression=NULL** or **expression<>(!=)NULL**, because **NULL** represents an unknown value, and these expressions cannot determine whether two unknown values are equal.

Examples

```
SELECT 2 BETWEEN 1 AND 3 AS RESULT;  
result  
-----  
t  
(1 row)  
  
SELECT 2 >= 1 AND 2 <= 3 AS RESULT;  
result  
-----  
t  
(1 row)  
  
SELECT 2 NOT BETWEEN 1 AND 3 AS RESULT;  
result  
-----  
f  
(1 row)  
  
SELECT 2 < 1 OR 2 > 3 AS RESULT;  
result  
-----  
f  
(1 row)  
  
SELECT 2+2 IS NULL AS RESULT;  
result  
-----  
f  
(1 row)  
  
SELECT 2+2 IS NOT NULL AS RESULT;  
result  
-----  
t  
(1 row)  
  
SELECT 2+2 ISNULL AS RESULT;  
result  
-----  
f  
(1 row)  
  
SELECT 2+2 NOTNULL AS RESULT;  
result
```

```
-----  
t  
(1 row)  
  
SELECT 2+2 IS DISTINCT FROM NULL AS RESULT;  
result  
-----  
t  
(1 row)  
  
SELECT 2+2 IS NOT DISTINCT FROM NULL AS RESULT;  
result  
-----  
f  
(1 row)
```

21.8.2 Conditional Expressions

Data that meets the requirements specified by conditional expressions are filtered during SQL statement execution.

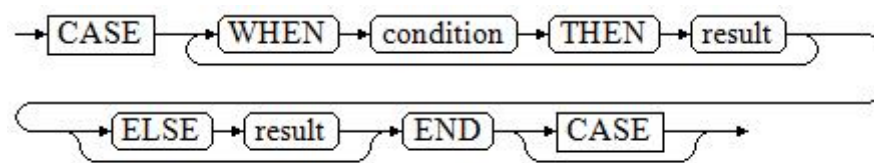
Conditional expressions include the following types:

- CASE

CASE expressions are similar to the **CASE** statements in other coding languages.

Figure 21-3 shows the syntax of a **CASE** expression.

Figure 21-3 case::=



A **CASE** clause can be used in a valid expression. **condition** is an expression that returns a value of Boolean type.

- If the result is **true**, the result of the **CASE** expression is the required result.
- If the result is false, the following **WHEN** or **ELSE** clauses are processed in the same way.
- If every **WHEN condition** is false, the result of the expression is the result of the **ELSE** clause. If the **ELSE** clause is omitted and has no match condition, the result is NULL.

CAUTION

- Keep the number of nested **case when** layers at 3 or fewer. The parser must expand all related fields during parsing. Deep recursion can lead to high memory usage.
- For complex **case when** statements, use a subquery instead. For details, see [Subquery Expressions](#).

Example:

```
CREATE TABLE tpcds.case_when_t1(CW_COL1 INT) DISTRIBUTE BY HASH (CW_COL1);

INSERT INTO tpcds.case_when_t1 VALUES (1), (2), (3);

SELECT * FROM tpcds.case_when_t1;
cw_col1
-----
      3
      1
      2
(3 rows)

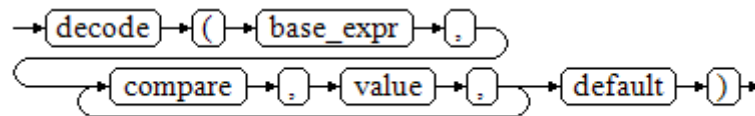
SELECT CW_COL1, CASE WHEN CW_COL1=1 THEN 'one' WHEN CW_COL1=2 THEN 'two' ELSE 'other'
END FROM tpcds.case_when_t1;
cw_col1 | case
-----+-----
      3 | other
      1 | one
      2 | two
(3 rows)

DROP TABLE tpcds.case_when_t1;
```

- **DECODE**

[Figure 21-4](#) shows the syntax of a **DECODE** expression.

Figure 21-4 decode::=



Compare each following **compare(n)** with **base_expr**, **value(n)** is returned if a **compare(n)** matches the **base_expr** expression. If **base_expr** does not match each **compare(n)**, the default value is returned.

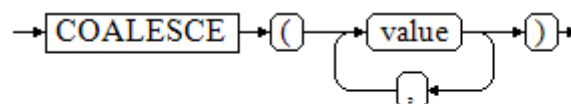
[Conditional Expression Functions](#) describes the examples.

```
SELECT DECODE('A','A',1,'B',2,0);
case
-----
      1
(1 row)
```

- **COALESCE**

[Figure 21-5](#) shows the syntax of a **COALESCE** expression.

Figure 21-5 coalesce::=



COALESCE returns its first non-NULL value. If all the arguments are NULL, return **NULL**. This value is replaced by the default value when data is displayed. Like a **CASE** expression, **COALESCE** only evaluates the parameters

that are needed to determine the result. That is, parameters to the right of the first non-null parameter are not evaluated.

Example:

```
CREATE TABLE tpcds.c_tabl(description varchar(10), short_description varchar(10), last_value
varchar(10))
DISTRIBUTE BY HASH (last_value);

INSERT INTO tpcds.c_tabl VALUES('abc', 'efg', '123');
INSERT INTO tpcds.c_tabl VALUES(NULL, 'efg', '123');

INSERT INTO tpcds.c_tabl VALUES(NULL, NULL, '123');

SELECT description, short_description, last_value, COALESCE(description, short_description, last_value)
FROM tpcds.c_tabl ORDER BY 1, 2, 3, 4;
description | short_description | last_value | coalesce
-----+-----+-----+-----
abc         | efg              | 123        | abc
           | efg              | 123        | efg
           |                  | 123        | 123
(3 rows)

DROP TABLE tpcds.c_tabl;
```

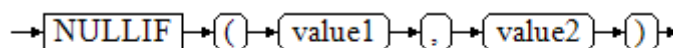
If **description** is not **NULL**, the value of **description** is returned. Otherwise, parameter **short_description** is calculated. If **short_description** is not **NULL**, the value of **short_description** is returned. Otherwise, parameter **last_value** is calculated. If **last_value** is not **NULL**, the value of **last_value** is returned. Otherwise, **none** is returned.

```
SELECT COALESCE(NULL, 'Hello World');
coalesce
-----
Hello World
(1 row)
```

- **NULLIF**

Figure 21-6 shows the syntax of a **NULLIF** expression.

Figure 21-6 nullif::=



Only if **value1** is equal to **value2** can **NULLIF** return the **NULL** value. Otherwise, **value1** is returned.

Example:

```
CREATE TABLE tpcds.null_if_t1 (
  NI_VALUE1 VARCHAR(10),
  NI_VALUE2 VARCHAR(10)
) DISTRIBUTE BY HASH (NI_VALUE1);

INSERT INTO tpcds.null_if_t1 VALUES('abc', 'abc');
INSERT INTO tpcds.null_if_t1 VALUES('abc', 'efg');

SELECT NI_VALUE1, NI_VALUE2, NULLIF(NI_VALUE1, NI_VALUE2) FROM tpcds.null_if_t1 ORDER BY 1,
2, 3;
ni_value1 | ni_value2 | nullif
-----+-----+-----
abc       | abc       |
abc       | efg       | abc
```

(2 rows)

```
DROP TABLE tpcds.null_if_t1;
```

If **value1** is equal to **value2**, **NULL** is returned. Otherwise, **value1** is returned.

```
SELECT NULLIF('Hello','Hello World');
```

```
nullif
```

```
-----
```

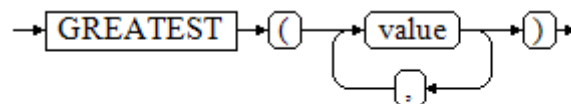
```
Hello
```

```
(1 row)
```

- GREATEST (maximum value) and LEAST (minimum value)

Figure 21-7 shows the syntax of a **GREATEST** expression.

Figure 21-7 greatest::=



You can select the maximum value from any numerical expression list.

```
SELECT greatest(9000,155555,2.01);
```

```
greatest
```

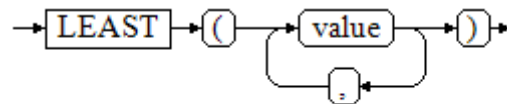
```
-----
```

```
155555
```

```
(1 row)
```

Figure 21-8 shows the syntax of a **LEAST** expression.

Figure 21-8 least::=



You can select the minimum value from any numerical expression list.

Each of the preceding numeric expressions can be converted into a common data type, which will be the data type of the result.

The NULL values in the list will be ignored. The result is **NULL** only if the results of all expressions are **NULL**.

```
SELECT least(9000,2);
```

```
least
```

```
-----
```

```
2
```

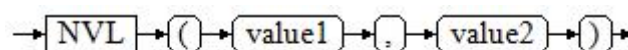
```
(1 row)
```

Conditional Expression Functions describes the examples.

- NVL

Figure 21-9 shows the syntax of an **NVL** expression.

Figure 21-9 nvl::=



If the value of **value1** is **NULL**, **value2** is returned. Otherwise, **value1** is returned.

Example:

```
SELECT nvl(null,1);
nvl
-----
1
(1 row)
SELECT nvl ('Hello World' ,1);
nvl
-----
Hello World
(1 row)
```

- IF

Figure 21-10 shows the syntax of an **IF** expression.

Figure 21-10 if::=



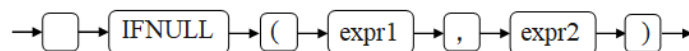
If the value of **bool_expr** is **true**, **expr1** is returned. Otherwise, **expr2** is returned.

Conditional Expression Functions describes the examples.

- IFNULL

Figure 21-11 shows the syntax of an **IFNULL** expression.

Figure 21-11 ifnull::=



It returns **expr1** or **expr2**. If **expr1** is not **NULL**, **expr1** is returned. Otherwise, **expr2** is returned.

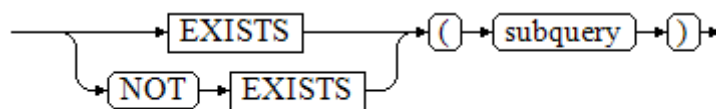
Conditional Expression Functions describes the examples.

21.8.3 Subquery Expressions

Subquery expressions include the following types:

- EXISTS/NOT EXISTS

Figure 21-12 shows the syntax of an **EXISTS/NOT EXISTS** expression.

Figure 21-12 EXISTS/NOT EXISTS::=

The parameter of an **EXISTS** expression is an arbitrary **SELECT** statement, or subquery. The subquery is evaluated to determine whether it returns any rows. If it returns at least one row, the result of **EXISTS** is "true". If the subquery returns no rows, the result of **EXISTS** is "false".

The subquery will generally only be executed long enough to determine whether at least one row is returned, not all the way to completion.

Example:

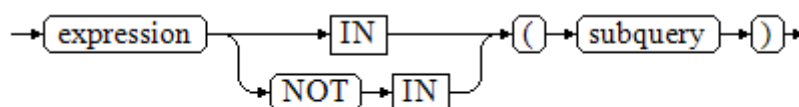
```
SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE EXISTS (SELECT d_dom FROM  
tpcds.date_dim WHERE d_dom = store_returns.sr_reason_sk and sr_customer_sk <10);  
sr_reason_sk | sr_customer_sk
```

| | |
|----|---|
| 13 | 2 |
| 22 | 5 |
| 17 | 7 |
| 25 | 7 |
| 3 | 7 |
| 31 | 5 |
| 7 | 7 |
| 14 | 6 |
| 20 | 4 |
| 5 | 6 |
| 10 | 3 |
| 1 | 5 |
| 15 | 2 |
| 4 | 1 |
| 26 | 3 |

(15 rows)

- IN/NOT IN

Figure 21-13 shows the syntax of an **IN/NOT IN** expression.

Figure 21-13 IN/NOT IN::=

The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result. The result of **IN** is "true" if any equal subquery row is found. The result is "false" if no equal row is found (including the case where the subquery returns no rows).

This is in accordance with SQL's normal rules for Boolean combinations of null values. If the columns corresponding to two rows equal and are not empty, the two rows are equal to each other. If any columns corresponding to the two rows do not equal and are not empty, the two rows are not equal to each other. Otherwise, the result is **NULL**. If there are no equal right-hand

values and at least one right-hand row yields null, the result of **IN** will be null, not false.

Example:

```
SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE sr_customer_sk IN (SELECT  
d_dom FROM tpcds.date_dim WHERE d_dom < 10);
```

sr_reason_sk | sr_customer_sk

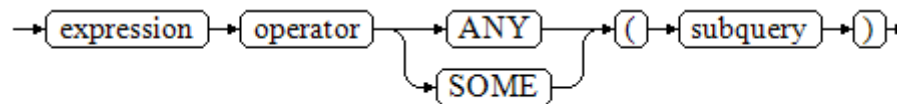
| -----+----- | |
|-------------|---|
| 10 | 3 |
| 26 | 3 |
| 22 | 5 |
| 31 | 5 |
| 1 | 5 |
| 32 | 5 |
| 32 | 5 |
| 4 | 1 |
| 15 | 2 |
| 13 | 2 |
| 33 | 4 |
| 20 | 4 |
| 33 | 8 |
| 5 | 6 |
| 14 | 6 |
| 17 | 7 |
| 3 | 7 |
| 25 | 7 |
| 7 | 7 |

(19 rows)

- ANY/SOME

Figure 21-14 shows the syntax of an **ANY/SOME** expression.

Figure 21-14 any/some::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of **ANY** is "true" if any true result is obtained. The result is "false" if no true result is found (including the case where the subquery returns no rows). **SOME** is a synonym of **ANY**. **IN** can be equivalently replaced with **ANY**.

Example:

```
SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE sr_customer_sk < ANY  
(SELECT d_dom FROM tpcds.date_dim WHERE d_dom < 10);
```

sr_reason_sk | sr_customer_sk

| -----+----- | |
|-------------|---|
| 26 | 3 |
| 17 | 7 |
| 32 | 5 |
| 32 | 5 |
| 13 | 2 |
| 31 | 5 |
| 25 | 7 |
| 5 | 6 |
| 7 | 7 |
| 10 | 3 |
| 1 | 5 |

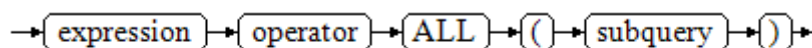
| | |
|----|---|
| 14 | 6 |
| 4 | 1 |
| 3 | 7 |
| 22 | 5 |
| 33 | 4 |
| 20 | 4 |
| 33 | 8 |
| 15 | 2 |

(19 rows)

- ALL

Figure 21-15 shows the syntax of an **ALL** expression.

Figure 21-15 all::=



The right-hand side is a parenthesized subquery, which must return exactly one column. The left-hand expression is evaluated and compared to each row of the subquery result using the given operator, which must yield a Boolean result. The result of **ALL** is "true" if all rows yield true (including the case where the subquery returns no rows). The result is "false" if any false result is found.

Example:

```
SELECT sr_reason_sk,sr_customer_sk FROM tpcds.store_returns WHERE sr_customer_sk < all(SELECT
d_dom FROM tpcds.date_dim WHERE d_dom < 10);
sr_reason_sk | sr_customer_sk
-----+-----
(0 rows)
```

21.8.4 Array Expressions

IN

expression **IN** (*value* [, ...])

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list meets the expression result on the left, the result of **IN** is **true**. If no result meets the requirements, the result of **IN** is **false**.

Example:

```
SELECT 8000+500 IN (10000, 9000) AS RESULT;
result
-----
f
(1 row)
```

NOTE

If the expression result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of **IN** is **null** rather than **false**. This method is consistent with the Boolean rules used when SQL statements return empty values.

NOT IN

expression **NOT IN** (*value* [, ...])

The parentheses on the right contain an expression list. The expression result on the left is compared with the content in the expression list. If the content in the list does not meet the expression result on the left, the result of **NOT IN** is **true**. If any content meets the expression result, the result of **NOT IN** is **false**.

Example:

```
SELECT 8000+500 NOT IN (10000, 9000) AS RESULT;  
result  
-----  
t  
(1 row)
```

NOTE

If the query statement result is null or the expression list does not meet the expression conditions and at least one empty value is returned for the expression list on the right, the result of **NOT IN** is **null** rather than **false**. This method is consistent with the Boolean rules used when SQL statements return empty values.

In all situations, **X NOT IN Y** equals to **NOT(X IN Y)**.

ANY/SOME (array)

expression operator **ANY** (*array expression*)

expression operator **SOME** (*array expression*)

```
SELECT 8000+500 < SOME (array[10000,9000]) AS RESULT;  
result  
-----  
t  
(1 row)  
SELECT 8000+500 < ANY (array[10000,9000]) AS RESULT;  
result  
-----  
t  
(1 row)
```

The parentheses on the right contain an array expression, which must generate an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

- If at least one comparison result is true, the result of **ANY** is **true**.
- If no comparison result is true, the result of **ANY** is false.

NOTE

If no comparison result is true and the array expression generates at least one null value, the value of **ANY** is **NULL**, rather than false. This method is consistent with the Boolean rules used when SQL statements return empty values.

SOME is a synonym of **ANY**.

ALL (array)

expression operator **ALL** (*array expression*)

The parentheses on the right contain an array expression, which must generate an array value. The result of the expression on the left uses operators to compute and compare the results in each row of the array expression. The comparison result must be a Boolean value.

- The result of **ALL** is "true" if all comparisons yield **true** (including the case where the array has zero elements).
- The result is **false** if any false result is found.

If the array expression yields a null array, the result of **ALL** will be null. If the left-hand expression yields null, the result of **ALL** is ordinarily null (though a non-strict comparison operator could possibly yield a different result). Also, if the right-hand array contains any null elements and no false comparison result is obtained, the result of **ALL** will be null, not true (again, assuming a strict comparison operator). This method is consistent with the Boolean rules used when SQL statements return empty values.

```
SELECT 8000+500 < ALL (array[10000,9000]) AS RESULT;  
result  
-----  
t  
(1 row)
```

21.8.5 Row Expressions

Syntax:

row_constructor operator row_constructor

Both sides of the row expression are row constructors. The values of both rows must have the same number of fields and they are compared with each other. The row comparison allows operators including =, <>, <, <=, and >= or a similar operator.

The use of operators =<> is slightly different from other operators. If all fields of two rows are not empty and equal, the two rows are equal. If any field in two rows is not empty and not equal, the two rows are not equal. Otherwise, the comparison result is null.

For operators <, <=, >, and >=, the fields in rows are compared from left to right until a pair of fields that are not equal or are empty are detected. If the pair of fields contains at least one null value, the comparison result is null. Otherwise, the comparison result of this pair of fields is the final result.

Example:

```
SELECT ROW(1,2,NULL) < ROW(1,3,0) AS RESULT;  
result  
-----  
t  
(1 row)
```

21.9 Type Conversion

21.9.1 Overview

Context

SQL is a typed language. That is, every data item has an associated data type which determines its behavior and allowed usage. DWS has an extensible type system that is more general and flexible than other SQL implementations. Hence, most type conversion behavior in DWS is governed by general rules. This allows the use of mixed-type expressions.

The DWS scanner/parser divides lexical elements into five fundamental categories: integers, floating-point numbers, strings, identifiers, and keywords. Constants of most non-numeric types are first classified as strings. The SQL language definition allows specifying type names with constant strings. For example, the query:

```
SELECT text 'Origin' AS "label", point '(0,0)' AS "value";
label | value
-----+-----
Origin | (0,0)
(1 row)
```

has two literal constants, of type **text** and **point**. If a type is not specified for a string literal, then the placeholder type **unknown** is assigned initially.

There are four fundamental SQL constructs requiring distinct type conversion rules in the DWS parser:

- **Function calls**
Much of the SQL type system is built around a rich set of functions. Functions can have one or more arguments. Since SQL permits function overloading, the function name alone does not uniquely identify the function to be called. The parser must select the right function based on the data types of the supplied arguments.
- **Operators**
SQL allows expressions with prefix and postfix unary (one-argument) operators, as well as binary (two-argument) operators. Like functions, operators can be overloaded, so the same problem of selecting the right operator exists.
- **Value Storage**
SQL **INSERT** and **UPDATE** statements place the results of expressions into a table. The expressions in the statement must be matched up with, and perhaps converted to, the types of the target columns.
- **UNION, CASE, and related constructs**
Since all query results from a unionized **SELECT** statement must appear in a single set of columns, the types of the results of each **SELECT** clause must be matched up and converted to a uniform set. Similarly, the result expressions of a **CASE** construct must be converted to a common type so that the **CASE** expression as a whole has a known output type. The same holds for **ARRAY** constructs, and for the **GREATEST** and **LEAST** functions.

The system catalog **PG_CAST** stores information about the conversion between data types and how to perform the conversion. For details, see section "PG_CAST" in the *Data Warehouse Service (DWS) Developer Guide*.

The return type and conversion behavior of an expression are determined during semantic analysis. Data types are divided into several basic type categories, including **boolean**, **numeric**, **string**, **bitstring**, **datetime**, **timespan**, **geometric**, and **network**. Within each category there can be one or more preferred types, which are preferred when there is a choice of possible types. With careful selection of preferred types and available implicit casts, it is possible to ensure that ambiguous expressions (those with multiple candidate parsing solutions) can be resolved in a useful way.

All type conversion rules are designed based on the following principles:

- Implicit conversions should never have surprising or unpredictable outcomes.
- There should be no extra overhead in the parser or executor if a query does not need implicit type conversion. That is, if a query is well-formed and the types already match, then the query should execute without spending extra time in the parser and without introducing unnecessary implicit conversion calls in the query.
- Additionally, if a query usually requires an implicit conversion for a function, and if then the user defines a new function with the correct argument types, the parser should use this new function.

Converting Empty Strings to Numeric Values in TD-Compatible Mode

- Different from the Oracle database, which processes an empty string as NULL, Teradata database converts an empty string to **0** by default. Therefore, when an empty string is queried, value **0** is found. Similarly, in TD-compatible mode, the empty string is converted to **0** of the corresponding numeric type by default. In addition, '-', '+', and ' ' are converted to **0** by default in TD-compatible mode, but an error is reported for a decimal point string. Example:

```
create table t1(no int,col varchar);
insert into t1 values(1,'');
insert into t1 values(2,null);
select * from t1 where col is null;
no | col
----+-----
2 | 
(1 row)
select * from t1 where col="";
no | col
----+-----
1 | 
(1 row)
```
- The method of converting an empty string into a numeric value in MySQL-compatible mode is the same as that in TD-compatible mode.

21.9.2 Operators

Operator Type Resolution

1. Select the operators to be considered from the **pg_operator** system catalog. Considered operators are those with the matching name and argument count. If the search path finds multiple available operators, only the most suitable one is considered.
2. Look for the best match.
 - a. Discard candidate operators for which the input types do not match and cannot be converted (using an implicit conversion) to match. **unknown**

literals are assumed to be convertible to anything for this purpose. If only one candidate remains, use it; else continue to the next step.

- b. Run through all candidates and keep those with the most exact matches on input types. Domains are considered the same as their base type for this purpose. Keep all candidates if there are no exact matches. If only one candidate remains, use it; else continue to the next step.
- c. Review all available candidate operators and select the one that supports the most preferred position types, which fall under the input data type's category, when type conversion is needed. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.
- d. If any input arguments are of **unknown** types, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Now discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.
- e. If there are both **unknown** and known-type arguments, and all the known-type arguments have the same type, assume that the **unknown** arguments are also of that type, and check which candidates can accept that type at the unknown-argument positions. If exactly one candidate passes this test, use it. Otherwise, an error is reported.

Examples

Example 1: factorial operator type resolution. There is only one factorial operator (postfix !) defined in the system catalog, and it takes an argument of type **bigint**. The scanner assigns an initial type of **bigint** to the argument in this query expression:

```
SELECT 40 ! AS "40 factorial";
      40 factorial
-----
8159152832478977343456112695961158942720000000000
(1 row)
```

So the parser does a type conversion on the operand and the query is equivalent to:

```
SELECT CAST(40 AS bigint) ! AS "40 factorial";
```

Example 2: string concatenation operator type resolution. A string-like syntax is used for working with string types and for working with complex extension types. Strings with unspecified type are matched with likely operator candidates. An example with one unspecified argument:

```
SELECT text 'abc' || 'def' AS "text and unknown";
text and unknown
-----
```

```
abcdef  
(1 row)
```

In this example, the parser looks for an operator whose parameters are of the text type. Such an operator is found.

Here is a concatenation of two values of unspecified types:

```
SELECT 'abc' || 'def' AS "unspecified";  
unspecified  
-----  
abcdef  
(1 row)
```

NOTE

In this case there is no initial hint for which type to use, since no types are specified in the query. So, the parser looks for all candidate operators and finds that there are candidates accepting both string-category and bit-string-category inputs. Since string category is preferred when available, that category is selected, and then the preferred type for strings, **text**, is used as the specific type to resolve the unknown-type literals.

Example 3: absolute-value and negation operator type resolution. The DWS operator catalog has several entries for the prefix operator @. All the entries implement absolute-value operations for various numeric data types. One of these entries is for type **float8**, which is the preferred type in the numeric category. Therefore, DWS will use that entry when faced with an **unknown** input:

```
SELECT @ '-4.5' AS "abs";  
abs  
-----  
4.5  
(1 row)
```

Here the system has implicitly resolved the unknown-type literal as type **float8** before applying the chosen operator.

Example 4: array inclusion operator type resolution. The following is an example of resolving an operator with one known and one unknown input:

```
SELECT array[1,2] <@ '{1,2,3}' as "is subset";  
is subset  
-----  
t  
(1 row)
```

NOTE

In the **pg_operator** table of DWS, several entries correspond to the infix operator <@, but the only two that may accept an integer array on the left-hand side are array inclusion (**anyarray <@ anyarray**) and range inclusion (**anyelement <@ anyrange**). Because none of these polymorphic pseudo-types (see [Pseudo-Types](#)) is considered preferred, the parser cannot resolve the ambiguity on that basis. However, [2.e](#) tells it to assume that the unknown-type literal is of the same type as the other input, that is, integer array. Now only one of the two operators can match, so array inclusion is selected. (If you select range inclusion, an error will be reported because the string does not have the right format to be a range literal.)

21.9.3 Functions

Function Type Resolution

1. Select the functions to be considered from the **pg_proc** system catalog. If a non-schema-qualified function name was used, the functions in the current

search path are considered. If a qualified function name was given, only functions in the specified schema are considered.

If the search path finds multiple functions of different argument types, a proper function in the path is considered.

2. Check for a function accepting exactly the input argument types. If the function exists, use it. Cases involving **unknown** will never find a match at this step.
3. If no exact match is found, see if the function call appears to be a special type conversion request.
4. Look for the best match.
 - a. Discard candidate functions for which the input types do not match and cannot be converted (using an implicit conversion) to match. **unknown** literals are assumed to be convertible to anything for this purpose. If only one candidate remains, use it; else continue to the next step.
 - b. Run through all candidates and keep those with the most exact matches on input types. Domains are considered the same as their base type for this purpose. Keep all candidates if none has exact matches. If only one candidate remains, use it; else continue to the next step.
 - c. Run through all candidates and keep those that accept preferred types at the most positions where type conversion will be required. Keep all candidates if none accepts preferred types. If only one candidate remains, use it; else continue to the next step.
 - d. If any input arguments are of **unknown** types, check the type categories accepted at those argument positions by the remaining candidates. At each position, select the string category if any candidate accepts that category. (This bias towards string is appropriate since an unknown-type literal looks like a string.) Otherwise, if all the remaining candidates accept the same type category, select that category; otherwise fail because the correct choice cannot be deduced without more clues. Now discard candidates that do not accept the selected type category. Furthermore, if any candidate accepts a preferred type in that category, discard candidates that accept non-preferred types for that argument. Keep all candidates if none survives these tests. If only one candidate remains, use it; else continue to the next step.
 - e. If there are both **unknown** and known-type arguments, and all the known-type arguments have the same type, assume that the **unknown** arguments are also of that type, and check which candidates can accept that type at the **unknown**-argument positions. If exactly one candidate passes this test, use it. Otherwise, fail.

Examples

Example 1: Use the rounding function argument type resolution as the first example. There is only one **round** function that takes two arguments; it takes a first argument of type **numeric** and a second argument of type **integer**. So the following query automatically converts the first argument of type **integer** to **numeric**:

```
SELECT round(4, 4);
round
-----
```

```
4.0000  
(1 row)
```

That query is converted by the parser to:

```
SELECT round(CAST (4 AS numeric), 4);
```

Since numeric constants with decimal points are initially assigned the type **numeric**, the following query will require no type conversion and therefore might be slightly more efficient:

```
SELECT round(4.0, 4);
```

Example 2: Use the substring function type resolution as the second example. There are several **substr** functions, one of which takes types **text** and **integer**. If called with a string constant of unspecified type, the system chooses the candidate function that accepts an argument of the preferred category **string** (namely of type **text**).

```
SELECT substr('1234', 3);  
substr  
-----  
34  
(1 row)
```

If the string is declared to be of type **varchar**, as might be the case if it comes from a table, then the parser will try to convert it to become **text**:

```
SELECT substr(varchar '1234', 3);  
substr  
-----  
34  
(1 row)
```

This is transformed by the parser to effectively become:

```
SELECT substr(CAST (varchar '1234' AS text), 3);
```

NOTE

The parser learns from the **pg_cast** catalog that **text** and **varchar** are binary-compatible, meaning that one can be passed to a function that accepts the other without doing any physical conversion. Therefore, no type conversion is inserted in this case.

And, if the function is called with an argument of type **integer**, the parser will try to convert that to **text**:

```
SELECT substr(1234, 3);  
substr  
-----  
34  
(1 row)
```

This is transformed by the parser to effectively become:

```
SELECT substr(CAST (1234 AS text), 3);  
substr  
-----  
34  
(1 row)
```

21.9.4 Value Storage

Value Storage Type Resolution

1. Search for an exact match with the target column.
2. Try to convert the expression to the target type. This will succeed if there is a registered cast between the two types. If the expression is an unknown-type literal, the content of the literal string will be fed to the input conversion routine for the target type.
3. Check whether there is a sizing cast for the target type. A sizing cast is a cast from that type to itself. If one is found in the **pg_cast** catalog, apply it to the expression before storing into the destination column. The implementation function for such a cast always takes an extra parameter of type **integer**. The parameter receives the destination column's **atttypmod** value (typically its declared length, although the interpretation of **atttypmod** varies for different data types), and may take a third boolean parameter that says whether the cast is explicit or implicit. The cast function is responsible for applying any length-dependent semantics such as size checking or truncation.

Examples

Use the **character** storage type conversion as an example. For a target column declared as **character(20)** the following statement shows that the stored value is sized correctly:

```
CREATE TABLE x1
(
  customer_sk      integer,
  customer_id      char(20),
  first_name       char(6),
  last_name        char(8)
)
with (orientation = column,compression=middle)
distribute by hash (last_name);

INSERT INTO x1(customer_sk, customer_id, first_name) VALUES (3769, 'abcdef', 'Grace');

SELECT customer_id, octet_length(customer_id) FROM x1;
  customer_id | octet_length
-----+-----
  abcdef      |           20
(1 row)
DROP TABLE x1;
```

NOTE

What has really happened here is that the two unknown literals are resolved to **text** by default, allowing the || operator to be resolved as **text** concatenation. Then the **text** result of the operator is converted to **bpchar** ("blank-padded char", the internal name of the **character** data type) to match the target column type. Since the conversion from **text** to **bpchar** is binary-coercible, this conversion does not insert any real function call. Finally, the sizing function **bpchar(bpchar, integer, boolean)** is found in the system catalog and used for the operator's result and the stored column length. This type-specific function performs the required length check and addition of padding spaces.

21.9.5 UNION, CASE, and Related Constructs

SQL UNION constructs must match up possibly dissimilar types to become a single result set. Since all query results from a **SELECT UNION** statement must appear in

a single set of columns, the types of the results of each **SELECT** clause must be matched up and converted to a uniform set. Similarly, the result expressions of a **CASE** construct must be converted to a common type so that the **CASE** expression as a whole has a known output type. The same holds for **ARRAY** constructs, and for the **GREATEST** and **LEAST** functions.

Type Resolution for UNION, CASE, and Related Constructs

- If all inputs are of the same type, and it is not unknown, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text** (the preferred type of the string category). Otherwise, **unknown** inputs are ignored.
- If the entered values are not the same type, the query fails (except for the **unknown** type).
- When input types fall under the same category, the preferred type of that category is chosen, unless the union operation selects the type of the first branch as the chosen type.

NOTE

typcategory in the **pg_type** system catalog indicates the data type category.
typispreferred indicates whether a type is preferred in **typcategory**.

- All the input is converted to the selected type. (The original length of a string is retained). Fail if there is not an implicit conversion from a given input to the selected type.
- If the input contains the json, txid_snapshot, sys_refcursor, or geometry type, **UNION** cannot be performed.

Type Resolution for CASE, COALESCE, IF, and IFNULL in TD-Compatible Mode

- If all inputs are of the same type, and it is not unknown, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text**.
- If inputs are of string type (including **unknown** which is resolved as type **text**) and digit type, resolve as the string type. If the inputs are not of the two types, fail.
- If the non-unknown inputs are all of the same type category, choose the input type which is a preferred type in that category, if there is one.
- Convert all inputs to the selected type. Fail if there is not an implicit conversion from a given input to the selected type.

Type Resolution for CASE, COALESCE, IF, and IFNULL in MySQL-Compatible Mode

- If all inputs are of the same type, and it is not unknown, resolve as that type.
- If all inputs are of type **unknown**, resolve as type **text**.
- If some inputs are of type **unknown** and the others are of a non-**unknown** type, resolve as that non-**unknown** type.
- If the inputs are of different non-**unknown** types, treat type **enum** as type **text** for comparison.
- If the non-**unknown** inputs are all of the same type, choose a preferred type, if there is one. If the inputs are of different types, resolve as type **text**.

- Convert all inputs to the selected type. Fail if there is not an implicit conversion from a given input to the selected type.

Examples

Example 1: Use type resolution with unknown types in a union as the first example. Here, the unknown-type literal **'b'** will be resolved to type **text**.

```
SELECT text 'a' AS "text" UNION SELECT 'b';
text
-----
a
b
(2 rows)
```

Example 2: Use type resolution in a simple union as the second example. The literal **1.2** is of type **numeric**, and the **integer** value **1** can be cast implicitly to **numeric**, so that type is used.

```
SELECT 1.2 AS "numeric" UNION SELECT 1;
numeric
-----
1
1.2
(2 rows)
```

Example 3: Use type resolution in a transposed union as the third example. Here, since type **real** cannot be implicitly cast to **integer**, but **integer** can be implicitly cast to **real**, the union result type is resolved as **real**.

```
SELECT 1 AS "real" UNION SELECT CAST('2.2' AS REAL);
real
-----
1
2.2
(2 rows)
```

Example 4: Use type resolution in the **COALESCE** function with input values of types **int** and **varchar** as the fourth example. Type resolution fails in ORA-compatible mode. The types are resolved as type **varchar** in TD-compatible mode, and as type **text** in MySQL-compatible mode.

```
-- Create the ora_db, td_db, and mysql_db databases by setting dbcompatibility to ORA, TD, and MySQL,
-- respectively:
CREATE DATABASE ora_db dbcompatibility = 'ORA';
CREATE DATABASE td_db dbcompatibility = 'TD';
CREATE DATABASE mysql_db dbcompatibility = 'MySQL';

-- Switch to the ora_db database:
\c ora_db

-- Create the t1 table:
ora_db=# CREATE TABLE t1(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for
-- COALESCE:
ora_db=# EXPLAIN SELECT coalesce(a, b) FROM t1;
ERROR: COALESCE types integer and character varying cannot be matched
CONTEXT: referenced column: coalesce

-- Delete the table:
ora_db=# DROP TABLE t1;

-- Switch to the td_db database:
ora_db=# \c td_db
```

```
-- Create the t2 table:
td_db=# CREATE TABLE t2(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for COALESCE:
td_db=# EXPLAIN VERBOSE select coalesce(a, b) from t2;
          QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+-----
 1 | -> Data Node Scan on "__REMOTE_FQS_QUERY__" |      0 |           |         | 0 | 0.00

          Targetlist Information (identified by plan id)
-----
 1 --Data Node Scan on "__REMOTE_FQS_QUERY__"
    Output: (COALESCE((t2.a)::character varying, t2.b))
    Node/s: All datanodes
    Remote query: SELECT COALESCE(a::character varying, b) AS "coalesce" FROM public.t2
(10 rows)

-- Delete the table:
td_db=# DROP TABLE t2;

-- Switch to the mysql_db database:
td_db=# \c mysql_db

-- Create the t3 table:
mysql_db=# CREATE TABLE t3(a int, b varchar(10));

-- Show the execution plan of a statement for querying the types int and varchar of input parameters for COALESCE:
mysql_db=# EXPLAIN VERBOSE select coalesce(a, b) from t3;
          QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+-----
 1 | -> Data Node Scan on "__REMOTE_FQS_QUERY__" |      0 |           |         | 0 | 0.00

          Targetlist Information (identified by plan id)
-----
 1 --Data Node Scan on "__REMOTE_FQS_QUERY__"
    Output: (COALESCE((t3.a)::text, (t3.b)::text))
    Node/s: All datanodes
    Remote query: SELECT COALESCE(a::text, b::text) AS "coalesce" FROM public.t3
(10 rows)

-- Delete the table:
mysql_db=# DROP TABLE t3;

-- Switch to the gaussdb database.
mysql_db=# \c gaussdb

-- Delete the databases:
DROP DATABASE ora_db;
DROP DATABASE td_db;
DROP DATABASE mysql_db;
```

21.10 Full Text Search

21.10.1 Introduction

21.10.1.1 Full-Text Retrieval

Textual search operators have been used in databases for years. The DWS has `~`, `~*`, `LIKE`, and `ILIKE` operators for textual data types, but they lack many essential properties required by modern information systems. This problem can be solved by using indexes and dictionaries.

Text search lacks the following essential properties required by information systems:

- There is no linguistic support, even for English.
Regular expressions are not sufficient because they cannot easily handle derived words. For example, you might miss documents that contain **satisfies**, although you probably would like to find them when searching for **satisfy**. It is possible to use **OR** to search for multiple derived forms, but this is tedious and error-prone, because some words can have several thousand derivatives.
- They provide no ordering (ranking) of search results, which makes them ineffective when thousands of matching documents are found.
- They tend to be slow because there is no index support, so they must process all documents for every search.

Full text indexing allows documents to be preprocessed and an index is saved for later rapid searching. Preprocessing includes:

- Parsing documents into tokens
It is useful to identify various classes of tokens, for example, numbers, words, complex words, and email addresses, so that they can be processed differently. In principle, token classes depend on the specific application, but for most purposes it is adequate to use a predefined set of classes.
- Converting tokens into lexemes
A lexeme is a string, just like a token, but it has been normalized so that different forms of the same word are made alike. For example, normalization almost always includes folding upper-case letters to lower-case, and often involves removal of suffixes (such as **s** or **es** in English). This allows searches to find variant forms of the same word, without tediously entering all the possible variants. Also, this step typically eliminates stop words, which are words that are so common that they are useless for searching. (In short, tokens are raw fragments of the document text, while lexemes are words that are believed useful for indexing and searching.) DWS uses dictionaries to perform this step and provides various standard dictionaries.
- Storing preprocessed documents optimized for searching
For example, each document can be represented as a sorted array of normalized lexemes. Along with the lexemes, it is often desirable to store positional information for proximity ranking. Therefore, a document that contains a more "dense" region of query words is assigned with a higher rank than the one with scattered query words.

Dictionaries allow fine-grained control over how tokens are normalized. With appropriate dictionaries, you can define stop words that should not be indexed.

A data type **tsvector** is provided for storing preprocessed documents, along with a type **tsquery** for storing query conditions. For details, see [Text Search Types](#). For details about the functions and operators provided for the **tsvector** data type, see

Text Search Functions and Operators. The matching operator @@ is the most important. For details, see [Basic Text Matching](#).

21.10.1.2 What Is a Document?

A document is the unit of searching in a full text search system; for example, a magazine article or email message. The text search engine must be able to parse documents and store associations of lexemes (keywords) with their parent document. Later, these associations are used to search for documents that contain query words.

For searches within DWS, a document is normally a textual column within a row of a database table, or possibly a combination (concatenation) of such columns, perhaps stored in several tables or obtained dynamically. In other words, a document can be constructed from different parts for indexing and it might not be stored anywhere as a whole. For example:

```
SELECT d_dow || '-' || d_dom || '-' || d_fy_week_seq AS identify_serials FROM tpeds.date_dim WHERE
d_fy_week_seq = 1;
identify_serials
-----
5-6-1
0-8-1
2-3-1
3-4-1
4-5-1
1-2-1
6-7-1
(7 rows)
```

NOTICE

Actually, in these example queries, **coalesce** should be used to prevent a single **NULL** attribute from causing a **NULL** result for the whole document.

Another possibility is to store the documents as simple text files in the file system. In this case, the database can be used to store the full text index and to execute searches, and some unique identifier can be used to retrieve the document from the file system. However, retrieving files from outside the database requires system administrator permissions or special function support, so this is less convenient than keeping all the data inside the database. Also, keeping everything inside the database allows easy access to document metadata to assist in indexing and display.

For text search purposes, each document must be reduced to the preprocessed **tsvector** format. Searching and relevance-based ranking are performed entirely on the **tsvector** representation of a document. The original text is retrieved only when the document has been selected for display to a user. We therefore often speak of the **tsvector** as being the document, but it is only a compact representation of the full document.

21.10.1.3 Basic Text Matching

Full text search in DWS is based on the match operator @@, which returns **true** if a **tsvector** (document) matches a **tsquery** (query). It does not matter which data type is written first:

```
SELECT 'a fat cat sat on a mat and ate a fat rat'::tsvector @@ 'cat & rat'::tsquery AS RESULT;
result
-----
t
(1 row)
SELECT 'fat & cow'::tsquery @@ 'a fat cat sat on a mat and ate a fat rat'::tsvector AS RESULT;
result
-----
f
(1 row)
```

As the above example suggests, a **tsquery** is not raw text, any more than a **tsvector** is. A **tsquery** contains search terms, which must be already-normalized lexemes, and may combine multiple terms using **AND**, **OR**, and **NOT** operators. For details, see [Text Search Types](#). There are functions **to_tsquery** and **plainto_tsquery** that are helpful in converting user-written text into a proper **tsquery**, for example by normalizing words appearing in the text. Similarly, **to_tsvector** is used to parse and normalize a document string. So in practice a text search match would look more like this:

```
SELECT to_tsvector('fat cats ate fat rats') @@ to_tsquery('fat & rat') AS RESULT;
result
-----
t
(1 row)
```

Observe that this match would not succeed if written as follows:

```
SELECT 'fat cats ate fat rats'::tsvector @@ to_tsquery('fat & rat') AS RESULT;
result
-----
f
(1 row)
```

In the preceding match, no normalization of the word **rats** will occur. Therefore, **rats** does not match **rat**.

The @@ operator also supports text input, allowing explicit conversion of a text string to **tsvector** or **tsquery** to be skipped in simple cases. The variants available are:

```
tsvector @@ tsquery
tsquery  @@ tsvector
text     @@ tsquery
text     @@ text
```

The form **text @@ tsquery** is equivalent to **to_tsvector(text) @@ tsquery**. The form **text @@ text** is equivalent to **to_tsvector(text) @@ plainto_tsquery(text)**.

21.10.1.4 Configurations

Full text search functionality includes the ability to do many more things: skip indexing certain words (stop words), process synonyms, and use sophisticated parsing, for example, parse based on more than just white space. This functionality is controlled by text search configurations. DWS comes with predefined configurations for many languages, and you can easily create your own configurations. (The **\dF** command of **gsql** shows all available configurations.)

During installation an appropriate configuration is selected and **default_text_search_config** is set accordingly in **postgresql.conf**. If you are using the same text search configuration for the entire cluster you can use the value in **postgresql.conf**. To use different configurations throughout the cluster but the

same configuration within any one database, use `ALTER DATABASE ... SET`. Otherwise, you can set **default_text_search_config** in each session.

Each text search function that depends on a configuration has an optional argument, so that the configuration to use can be specified explicitly. **default_text_search_config** is used only when this argument is omitted.

To make it easier to build custom text search configurations, a configuration is built up from simpler database objects. DWS's text search facility provides the following types of configuration-related database objects:

- Text search parsers break documents into tokens and classify each token (for example, as words or numbers).
- Text search dictionaries convert tokens to normalized form and reject stop words.
- Text search templates provide the functions underlying dictionaries. (A dictionary simply specifies a template and a set of parameters for the template.)
- Text search configurations select a parser and a set of dictionaries to use to normalize the tokens produced by the parser.

21.10.2 Tables and Indexes

21.10.2.1 Searching a Table

It is possible to do a full text search without an index.

- A simple query to print each row that contains the word **science** in its **body** column is as follows:

```
DROP SCHEMA IF EXISTS tsearch CASCADE;

CREATE SCHEMA tsearch;

CREATE TABLE tsearch.pgweb(id int, body text, title text, last_mod_date date);

INSERT INTO tsearch.pgweb VALUES(1, 'Philology is the study of words, especially the history and development of the words in a particular language or group of languages.', 'Philology', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(2, 'Mathematics is the science that deals with the logic of shape, quantity and arrangement.', 'Mathematics', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(3, 'Computer science is the study of processes that interact with data and that can be represented as data in the form of programs.', 'Computer science', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(4, 'Chemistry is the scientific discipline involved with elements and compounds composed of atoms, molecules and ions.', 'Chemistry', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(5, 'Geography is a field of science devoted to the study of the lands, features, inhabitants, and phenomena of the Earth and planets.', 'Geography', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(6, 'History is a subject studied in schools, colleges, and universities that deals with events that have happened in the past.', 'History', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(7, 'Medical science is the science of dealing with the maintenance of health and the prevention and treatment of disease.', 'Medical science', '2010-1-1');

INSERT INTO tsearch.pgweb VALUES(8, 'Physics is one of the most fundamental scientific disciplines, and its main goal is to understand how the universe behaves.', 'Physics', '2010-1-1');
```

```
SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector('english', body) @@
to_tsquery('english', 'science');
id | body | title
----+-----+-----
2 | Mathematics is the science that deals with the logic of shape, quantity and
    arrangement. | Mathematics
3 | Computer science is the study of processes that interact with data and that can be represented as
    data in the form of programs. | Computer science
5 | Geography is a field of science devoted to the study of the lands, features, inhabitants, and
    phenomena of the Earth and planets. | Geography
7 | Medical science is the science of dealing with the maintenance of health and the prevention and
    treatment of disease. | Medical science
(4 rows)
```

This will also find related words, such as **science**, since all these are reduced to the same normalized lexeme.

The query above specifies that the **english** configuration is to be used to parse and normalize the strings. Alternatively, you can omit the configuration parameters, and use the configuration set by **default_text_search_config**.

```
SHOW default_text_search_config;
default_text_search_config
-----
pg_catalog.english
(1 row)

SELECT id, body, title FROM tsearch.pgweb WHERE to_tsvector(body) @@ to_tsquery('science');
id | body | title
----+-----+-----
2 | Mathematics is the science that deals with the logic of shape, quantity and
    arrangement. | Mathematics
3 | Computer science is the study of processes that interact with data and that can be represented as
    data in the form of programs. | Computer science
5 | Geography is a field of science devoted to the study of the lands, features, inhabitants, and
    phenomena of the Earth and planets. | Geography
7 | Medical science is the science of dealing with the maintenance of health and the prevention and
    treatment of disease. | Medical science
(4 rows)
```

- A more complex example to select the ten most recent documents that contain **treatment** and **science** in the **title** or **body** column is as follows:

```
SELECT title FROM tsearch.pgweb WHERE to_tsvector(title || ' ' || body) @@ to_tsquery('treatment &
science') ORDER BY last_mod_date DESC LIMIT 10;
title
-----
Medical science
(1 rows)
```

For clarity we omitted the **coalesce** function calls which would be needed to find rows that contain **NULL** in one of the two columns.

The preceding examples show queries without using indexes. Most applications will find this approach too slow. Therefore, practical use of text searching usually requires creating an index, except perhaps for occasional ad-hoc searches.

21.10.2.2 Creating an Index

You can create a **GIN** index to speed up text searches:

```
CREATE INDEX pgweb_idx_1 ON tsearch.pgweb USING gin(to_tsvector('english', body));
```

The `to_tsvector()` function accepts one or two arguments.

If the one-argument version of the index is used, the system will use the configuration specified by **default_text_search_config** by default.

To create an index, the two-argument version must be used, or the index content may be inconsistent. Only text search functions that specify a configuration name can be used in expression indexes. Index content is not affected by **default_text_search_config**, because different entries could contain **tsvector**s that were created with different text search configurations, and there would be no way to guess which was which. It would be impossible to dump and restore such an index correctly.

Because the two-argument version of **to_tsvector** was used in the index above, only a query reference that uses the two-argument version of **to_tsvector** with the same configuration name will use that index. That is, **WHERE to_tsvector('english', body) @@ 'a & b'** can use the index, but **WHERE to_tsvector(body) @@ 'a & b'** cannot. This ensures that an index will be used only with the same configuration used to create the index entries.

More complex expression indexes can be set up when the configuration name of the index is specified by another column. For example:

```
CREATE INDEX pgweb_idx_2 ON tsearch.pgweb USING gin(to_tsvector('zhparser', body));
```

NOTE

In this example, `zhparser` supports only the UTF-8 or GBK database encoding format. If the `SQL_ASCII` encoding is used, an error will be reported.

where **body** is a column in the **pgweb** table. This allows mixed configurations in the same index while recording which configuration was used for each index entry. This would be useful, for example, if the document collection contained documents in different languages. Again, queries that are meant to use the index must be phrased to match, for example, **WHERE to_tsvector(config_name, body) @@ 'a & b'** must match **to_tsvector** in the index.

Indexes can even concatenate columns:

```
CREATE INDEX pgweb_idx_3 ON tsearch.pgweb USING gin(to_tsvector('english', title || ' ' || body));
```

Another approach is to create a separate **tsvector** column to hold the output of **to_tsvector**. This example is a concatenation of **title** and **body**, using **coalesce** to ensure that one column will still be indexed when the other is **NULL**:

```
ALTER TABLE tsearch.pgweb ADD COLUMN textsearchable_index_col tsvector;  
UPDATE tsearch.pgweb SET textsearchable_index_col = to_tsvector('english', coalesce(title, '') || ' ' || coalesce(body, ''));
```

Then, create a GIN index to speed up the search:

```
CREATE INDEX textsearch_idx_4 ON tsearch.pgweb USING gin(textsearchable_index_col);
```

Now you are ready to perform a fast full text search:

```
SELECT title  
FROM tsearch.pgweb  
WHERE textsearchable_index_col @@ to_tsquery('science & Computer')  
ORDER BY last_mod_date DESC  
LIMIT 10;
```

```
title
-----
Computer science
(1 rows)
```

One advantage of the separate-column approach over an expression index is that it is unnecessary to explicitly specify the text search configuration in queries in order to use the index. As shown in the preceding example, the query can depend on **default_text_search_config**. Another advantage is that searches will be faster, since it will not be necessary to redo the **to_tsvector** calls to verify index matches. The expression-index approach is simpler to set up, however, and it requires less disk space since the **tsvector** representation is not stored explicitly.

21.10.2.3 Constraints on Index Use

The following is an example of using an index:

```
create table table1 (c_int int,c_bigint bigint,c_varchar varchar,c_text text) with(orientation=row);

create text search configuration ts_conf_1(parser=POUND);
create text search configuration ts_conf_2(parser=POUND) with(split_flag='%');

set default_text_search_config='ts_conf_1';
create index idx1 on table1 using gin(to_tsvector(c_text));

set default_text_search_config='ts_conf_2';
create index idx2 on table1 using gin(to_tsvector(c_text));

select c_varchar,to_tsvector(c_varchar) from table1 where to_tsvector(c_text) @@ plainto_tsquery('#@.....
&**) and to_tsvector(c_text) @@ plainto_tsquery('company') and c_varchar is not null order by 1 desc limit
3;
```

In this example, **table1** has two GIN indexes created on the same column **c_text**, **idx1** and **idx2**, but these two indexes are created under different settings of **default_text_search_config**. Differences between this example and the scenario where one table has common indexes created on the same column are as follows:

- GIN indexes use different parsers (that is, different delimiters). In this case, the index data of **idx1** is different from that of **idx2**.
- In the specified scenario, the index data of multiple common indexes created on the same column is the same.

As a result, using **idx1** and **idx2** for the same query returns different results.

Constraints

Still use the above example. When:

- Multiple GIN indexes are created on the same column of the same table.
- The GIN indexes 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.

21.10.3 Controlling Text Search

21.10.3.1 Parsing Documents

DWS provides function **to_tsvector** for converting a document to the **tsvector** data type.

to_tsvector([*config regconfig*,] *document text*) returns **tsvector**

to_tsvector parses a textual document into tokens, reduces the tokens to lexemes, and returns a **tsvector**, which lists the lexemes together with their positions in the document. The document is processed according to the specified or default text search configuration. Here is a simple example:

```
SELECT to_tsvector('english', 'a fat cat sat on a mat - it ate a fat rats');
       to_tsvector
-----
'ate':9 'cat':3 'fat':2,11 'mat':7 'rat':12 'sat':4
```

In the preceding example we see that the resulting **tsvector** does not contain the words **a**, **on**, or **it**, the word **rats** became **rat**, and the punctuation sign (-) was ignored.

The **to_tsvector** function internally calls a parser which breaks the document text into tokens and assigns a type to each token. For each token, a list of dictionaries is consulted. where the list can vary depending on the token type. The first dictionary that recognizes the token emits one or more normalized lexemes to represent the token. For example:

- **rats** became **rat** because one of the dictionaries recognized that the word **rats** is a plural form of **rat**.
- Some words are recognized as stop words (see [Stop Words](#)), which causes them to be ignored since they occur too frequently to be useful in searching. In our example these are **a**, **on**, and **it**.
- If no dictionary in the list recognizes the token then it is also ignored. In this example that happened to the punctuation sign (-) because there are in fact no dictionaries assigned for its token type (**space symbols**), meaning space tokens will never be indexed.

The choices of parser, dictionaries and which types of tokens to index are determined by the selected text search configuration. It is possible to have many different configurations in the same database, and predefined configurations are available for various languages. In our example we used the default configuration **english** for the English language.

The function **setweight** can be used to label the entries of a **tsvector** with a given weight, where a weight is one of the letters **A**, **B**, **C**, or **D**. This is typically used to mark entries coming from different parts of a document, such as title versus body. Later, this information can be used for ranking of search results.

Because **to_tsvector(NULL)** will return **NULL**, you are advised to use **coalesce** whenever a column might be **NULL**. Here is the recommended method for creating a **tsvector** from a structured document:

```
CREATE TABLE tsearch.tt (id int, title text, keyword text, abstract text, body text, ti tsvector);

INSERT INTO tsearch.tt(id, title, keyword, abstract, body) VALUES (1, 'book', 'literature', 'Ancient poetry', 'Tang poem Song jambic verse');

UPDATE tsearch.tt SET ti =
    setweight(to_tsvector(coalesce(title, '')), 'A') ||
```



```
setweight(to_tsvector(coalesce(keyword,'')), 'B') ||  
setweight(to_tsvector(coalesce(abstract,'')), 'C') ||  
setweight(to_tsvector(coalesce(body,'')), 'D');  
DROP TABLE tsearch.tt;
```

Here we have used **setweight** to label the source of each lexeme in the finished **tsvector**, and then merged the labeled **tsvector** values using the **tsvector** concatenation operator **||**. For details about these operations, see [Manipulating tsvector](#).

21.10.3.2 Parsing Queries

DWS provides functions **to_tsquery** and **plainto_tsquery** for converting a query to the **tsquery** data type. **to_tsquery** offers access to more features than **plainto_tsquery**, but is less forgiving about its input.

to_tsquery([config regconfig,] querytext text) returns tsquery

to_tsquery creates a **tsquery** value from **querytext**, which must consist of single tokens separated by the Boolean operators **&** (AND), **|** (OR), and **!** (NOT). These operators can be grouped using parentheses. In other words, the input to **to_tsquery** must already follow the general rules for **tsquery** input, as described in [Text Search Types](#). The difference is that while basic **tsquery** input takes the tokens at face value, **to_tsquery** normalizes each token to a lexeme using the specified or default configuration, and discards any tokens that are stop words according to the configuration. For example:

```
SELECT to_tsquery('english', 'The & Fat & Rats');  
to_tsquery  
-----  
'fat' & 'rat'  
(1 row)
```

As in basic **tsquery** input, **weight(s)** can be attached to each lexeme to restrict it to match only **tsvector** lexemes of those **weight(s)**. For example:

```
SELECT to_tsquery('english', 'Fat | Rats:AB');  
to_tsquery  
-----  
'fat' | 'rat':AB  
(1 row)
```

Also, the asterisk (*****) can be attached to a lexeme to specify prefix matching:

```
SELECT to_tsquery('supern:*A & star:A*B');  
to_tsquery  
-----  
'supern':*A & 'star':*AB  
(1 row)
```

Such a lexeme will match any word having the specified string and weight in a **tsquery**.

plainto_tsquery([config regconfig,] querytext text) returns tsquery

plainto_tsquery transforms unformatted text **querytext** to **tsquery**. The text is parsed and normalized much as for **to_tsvector**, then the **&** (AND) Boolean operator is inserted between surviving words.

For example:

```
SELECT plainto_tsquery('english', 'The Fat Rats');  
plainto_tsquery  
-----
```

```
'fat' & 'rat'  
(1 row)
```

Note that **plainto_tsquery** cannot recognize Boolean operators, weight labels, or prefix-match labels in its input:

```
SELECT plainto_tsquery('english', 'The Fat & Rats:C');  
plainto_tsquery  
-----  
'fat' & 'rat' & 'c'  
(1 row)
```

Here, all the input punctuation was discarded as being space symbols.

21.10.3.3 Ranking Search Results

Ranking attempts to measure how relevant documents are to a particular query, so that when there are many matches the most relevant ones can be shown first. DWS provides two predefined ranking functions. They take into account lexical, proximity, and structural information; that is, they consider how often the query terms appear in the document, how close together the terms are in the document, and how important is the part of the document where they occur. However, the concept of relevancy is vague and application-specific. Different applications might require additional information for ranking, for example, document modification time. The built-in ranking functions are only examples. You can write your own ranking functions and/or combine their results with additional factors to fit your specific needs.

The two ranking functions currently available are:

```
ts_rank([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns float4
```

Ranks vectors based on the frequency of their matching lexemes.

```
ts_rank_cd([ weights float4[], ] vector tsvector, query tsquery [, normalization integer ]) returns float4
```

This function requires positional information in its input. Therefore, it will not work on "stripped" **tsvector** values. It will always return zero.

For both these functions, the optional **weights** argument offers the ability to weigh word instances more or less heavily depending on how they are labeled. The weight arrays specify how heavily to weigh each category of word, in the order:

```
{D-weight, C-weight, B-weight, A-weight}
```

If no **weights** are provided, then these defaults are used: {0.1, 0.2, 0.4, 1.0}

Typically weights are used to mark words from special areas of the document, like the title or an initial abstract, so they can be treated with more or less importance than words in the document body.

Since a longer document has a greater chance of containing a query term it is reasonable to take into account document size. For example, a hundred-word document with five instances of a search word is probably more relevant than a thousand-word document with five instances. Both ranking functions take an integer **normalization** option that specifies whether and how a document's length should impact its rank. The integer option controls several behaviors, so it is a bit mask: you can specify one or more behaviors using a vertical bar (|) (for example, 2|4).

- 0 (the default) ignores the document length
- 1 divides the rank by (1 + Logarithm of the document length)
- 2 divides the rank by the document length
- 4 divides the rank by the mean harmonic distance between extents (this is implemented only by ts_rank_cd)
- 8 divides the rank by the number of unique words in document
- 16 divides the rank by (1 + Logarithm of the number of unique words in document)
- 32 divides the rank by (itself + 1)

If more than one flag bit is specified, the transformations are applied in the order listed.

It is important to note that the ranking functions do not use any global information, so it is impossible to produce a fair normalization to 1% or 100% as sometimes desired. Normalization option 32 (**rank/(rank+1)**) can be used to scale all ranks into the range zero to one. This is just a cosmetic change, and it will not affect the ordering of the search results.

Here is an example that selects only the ten highest-ranked matches:

Run the following statements in a database that uses the UTF-8 or GBK encoding:

```
SELECT id, title, ts_rank_cd(to_tsvector(body), query) AS rank
FROM tsearch.pgweb, to_tsquery('science') query
WHERE query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
id | title | rank
---+-----+-----
11 | Philology | .2
2 | Mathematics | .1
12 | Geography | .1
13 | Computer science | .1
(4 rows)
```

This is the same example using normalized ranking:

```
SELECT id, title, ts_rank_cd(to_tsvector(body), query, 32 /* rank/(rank+1) */) AS rank
FROM tsearch.pgweb, to_tsquery('science') query
WHERE query @@ to_tsvector(body)
ORDER BY rank DESC
LIMIT 10;
id | title | rank
---+-----+-----
11 | Philology | .166667
2 | Mathematics | .0909091
12 | Geography | .0909091
13 | Computer science | .0909091
(4 rows)
```

The following example sorts query by Chinese word segmentation:

```
CREATE TABLE tsearch.ts_zhparser(id int, body text);
INSERT INTO tsearch.ts_zhparser VALUES (1, 'sort');
INSERT INTO tsearch.ts_zhparser VALUES(2, 'sort query');
INSERT INTO tsearch.ts_zhparser VALUES(3, 'query sort');
-- Accurate match
SELECT id, body, ts_rank_cd (to_tsvector('zhparser', body), query) AS rank FROM tsearch.ts_zhparser,
to_tsquery('sort') query WHERE query @@ to_tsvector (body);
id | body | rank
---+-----+-----
1 | sort | .1
```

```
(1 row)

-- Fuzzy match
SELECT id, body, ts_rank_cd (to_tsvector ('zhparser', body), query) AS rank FROM tsearch.ts_zhparser,
to_tsquery ('sort') query WHERE query @@ to_tsvector ('zhparser', body);
 id |  body  | rank 
-----+-----
  3 | query sort | .1
  1 | sort   | .1
  2 | sort query | .1
(3 rows)
```

Ranking can be expensive since it requires consulting the **tsvector** of each matching document, which can be I/O bound and therefore slow. Unfortunately, it is almost impossible to avoid since practical queries often result in large numbers of matches.

21.10.3.4 Highlighting Results

To present search results it is ideal to show a part of each document and how it is related to the query. Usually, search engines show fragments of the document with marked search terms. DWS provides function **ts_headline** that implements this functionality.

ts_headline([config regconfig,] document text, query tsquery [, options text]) returns text

ts_headline accepts a document along with a query, and returns an excerpt from the document in which terms from the query are highlighted. The configuration to be used to parse the document can be specified by **config**. If **config** is omitted, the **default_text_search_config** configuration is used.

If an options string is specified it must consist of a comma-separated list of one or more **option=value** pairs. The available options are:

- **StartSel, StopSel**: The strings with which to delimit query words appearing in the document, to distinguish them from other excerpted words. You must double-quote these strings if they contain spaces or commas.
- **MaxWords, MinWords**: These numbers determine the longest and shortest headlines to output.
- **ShortWord**: Words of this length or less will be dropped at the start and end of a headline. The default value of three eliminates common English articles.
- **HighlightAll**: Boolean flag. If **true** the whole document will be used as the headline, ignoring the preceding three parameters.
- **MaxFragments**: Maximum number of text excerpts or fragments to display. The default value of zero selects a non-fragment-oriented headline generation method. A value greater than zero selects fragment-based headline generation. This method finds text fragments with as many query words as possible and stretches those fragments around the query words. As a result query words are close to the middle of each fragment and have words on each side. Each fragment will be of at most **MaxWords** and words of length **ShortWord** or less are dropped at the start and end of each fragment. If not all query words are found in the document, then a single fragment of the first **MinWords** in the document will be displayed.
- **FragmentDelimiter**: When more than one fragment is displayed, the fragments will be separated by this string.

Any unspecified options receive these defaults:

```
StartSel=<b>, StopSel=</b>,  
MaxWords=35, MinWords=15, ShortWord=3, HighlightAll=FALSE,  
MaxFragments=0, FragmentDelimiter=" ... "
```

For example:

```
SELECT ts_headline('english',  
'The most common type of search  
is to find all documents containing given query terms  
and return them in order of their similarity to the  
query.',  
to_tsquery('english', 'query & similarity'));  
      ts_headline  
-----  
containing given <b>query</b> terms  
and return them in order of their <b>similarity</b> to the  
<b>query</b>.  
(1 row)  
  
SELECT ts_headline('english',  
'The most common type of search  
is to find all documents containing given query terms  
and return them in order of their similarity to the  
query.',  
to_tsquery('english', 'query & similarity'),  
'StartSel = <, StopSel = >');  
      ts_headline  
-----  
containing given <query> terms  
and return them in order of their <similarity> to the  
<query>.  
(1 row)
```

ts_headline uses the original document, not a **tsvector** summary, so it can be slow and should be used with care.

21.10.4 Additional Features

21.10.4.1 Manipulating tsvector

DWS provides functions and operators that can be used to manipulate documents that are already in **tsvector** type.

- **tsvector || tsvector**

The **tsvector** concatenation operator returns a new **tsvector** which combines the lexemes and positional information of the two **tsvectors** given as arguments. Positions and weight labels are retained during the concatenation. Positions appearing in the right-hand **tsvector** are offset by the largest position mentioned in the left-hand **tsvector**, so that the result is nearly equivalent to the result of performing **to_tsvector** on the concatenation of the two original document strings. (The equivalence is not exact, because any stop-words removed from the end of the left-hand argument will not affect the result, whereas they would have affected the positions of the lexemes in the right-hand argument if textual concatenation were used.)

One advantage of using concatenation in the **tsvector** form, rather than concatenating text before applying **to_tsvector**, is that you can use different configurations to parse different sections of the document. Also, because the **setweight** function marks all lexemes of the given **tsvector** the same way, it is necessary to parse the text and do **setweight** before concatenating if you want to label different parts of the document with different weights.

- `setweight(vector tsvector, weight "char")` returns `tsvector`
setweight returns a copy of the input `tsvector` in which every position has been labeled with the given weight, either **A**, **B**, **C**, or **D**. (**D** is the default for new `tsvectors` and as such is not displayed on output.) These labels are retained when `tsvectors` are concatenated, allowing words from different parts of a document to be weighted differently by ranking functions.

NOTICE

Note that weight labels apply to positions, not lexemes. If the input `tsvector` has been stripped of positions then **setweight** does nothing.

- `length(vector tsvector)` returns integer
Returns the number of lexemes stored in the vector.
- `strip(vector tsvector)` returns `tsvector`
Returns a `tsvector` which lists the same lexemes as the given `tsvector`, but which lacks any position or weight information. While the returned `tsvector` is much less useful than an unstripped `tsvector` for relevance ranking, it will usually be much smaller.

21.10.4.2 Manipulating Queries

DWS provides functions and operators that can be used to manipulate queries that are already in `tsquery` type.

- `tsquery && tsquery`
Returns the AND-combination of the two given `tsqueries`.
- `tsquery || tsquery`
Returns the OR-combination of the two given `tsqueries`.
- `!! tsquery`
Returns the negation (NOT) of the given `tsquery`.
- `numnode(query tsquery)` returns integer
Returns the number of nodes (lexemes plus operators) in a **tsquery**. This function is useful to determine if the query is meaningful (returns > 0), or contains only stop words (returns 0). For example:

```
SELECT numnode(plainto_tsquery('the any'));
NOTICE: text-search query contains only stop words or doesn't contain lexemes, ignored
CONTEXT: referenced column: numnode
numnode
-----
      0

SELECT numnode('foo & bar'::tsquery);
numnode
-----
      3
```

- `querytree(query tsquery)` returns text
Returns the portion of a **tsquery** that can be used for searching an index. This function is useful for detecting unindexable queries, for example those containing only stop words or only negated terms. For example:

```
SELECT querytree(to_tsquery('!defined'));
querytree
-----
T
(1 row)
```

21.10.4.3 Rewriting Queries

The **ts_rewrite** family of functions searches a given **tsquery** for occurrences of a target subquery, and replace each occurrence with a substitute subquery. In essence this operation is a **tsquery** specific version of substring replacement. A target and substitute combination can be thought of as a query rewrite rule. A collection of such rewrite rules can be a powerful search aid. For example, you can expand the search using synonyms (that is, new york, big apple, nyc, gotham) or narrow the search to direct the user to some hot topic.

- **ts_rewrite** (query tsquery, target tsquery, substitute tsquery) returns tsquery
This form of **ts_rewrite** simply applies a single rewrite rule: **target** is replaced by **substitute** wherever it appears in query. For example:

```
SELECT ts_rewrite('a & b':tsquery, 'a':tsquery, 'c':tsquery);
ts_rewrite
-----
'b' & 'c'
```

- **ts_rewrite** (query tsquery, select text) returns tsquery
This form of **ts_rewrite** accepts a starting query and a SQL select command, which is given as a text string. The **select** must yield two columns of **tsquery** type. For each row of the select result, occurrences of the first column value (the target) are replaced by the second column value (the substitute) within the current **query** value.

NOTE

Note that when multiple rewrite rules are applied in this way, the order of application can be important; so in practice you will want the source query to **ORDER BY** some ordering key.

Consider a real-life astronomical example. We will expand query supernovae using table-driven rewriting rules:

```
CREATE TABLE tsearch.aliaes (id int, t tsquery, s tsquery);

INSERT INTO tsearch.aliaes VALUES(1, to_tsquery('supernovae'), to_tsquery('supernovae|sn'));

SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT t, s FROM tsearch.aliaes');

ts_rewrite
-----
'crab' & ( 'supernova' | 'sn' )
```

We can change the rewriting rules just by updating the table:

```
UPDATE tsearch.aliaes
SET s = to_tsquery('supernovae|sn & !nebulae')
WHERE t = to_tsquery('supernovae');

SELECT ts_rewrite(to_tsquery('supernovae & crab'), 'SELECT t, s FROM tsearch.aliaes');

ts_rewrite
-----
'crab' & ( 'supernova' | 'sn' & '!nebula' )
```

Rewriting can be slow when there are many rewriting rules, since it checks every rule for a possible match. To filter out obvious non-candidate rules we

can use the containment operators for the **tsquery** type. In the example below, we select only those rules which might match the original query:

```
SELECT ts_rewrite('a & b'::tsquery, 'SELECT t,s FROM tsearch.aliaes WHERE "a & b"::tsquery @> t');
ts_rewrite
-----
'b' & 'a'
(1 row)
DROP TABLE ts_rewrite;
```

21.10.4.4 Gathering Document Statistics

The function **ts_stat** is useful for checking your configuration and for finding stop-word candidates.

```
ts_stat(sqlquery text, [ weights text, ]
      OUT word text, OUT ndoc integer,
      OUT nentry integer) returns setof record
```

sqlquery is a text value containing an SQL query which must return a single **tsvector** column. **ts_stat** executes the query and returns statistics about each distinct lexeme (word) contained in the **tsvector** data. The columns returned are:

- **word text**: the value of a lexeme.
- **ndoc integer**: number of documents (**tsvectors**) the word occurred in.
- **nentry integer**: total number of occurrences of the word.

If **weights** are supplied, only occurrences having one of those weights are counted. For example, to find the ten most frequent words in a document collection:

```
SELECT * FROM ts_stat('SELECT to_tsvector("english", sr_reason_sk) FROM tpccs.store_returns WHERE
sr_customer_sk < 10') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;;
 word | ndoc | nentry
-----+-----+-----
 32  |    2 |      2
 33  |    2 |      2
  1  |    1 |      1
 10  |    1 |      1
 13  |    1 |      1
 14  |    1 |      1
 15  |    1 |      1
 17  |    1 |      1
 20  |    1 |      1
 22  |    1 |      1
(10 rows)
```

The same, but counting only word occurrences with weight **A** or **B**:

```
SELECT * FROM ts_stat('SELECT to_tsvector("english", sr_reason_sk) FROM tpccs.store_returns WHERE
sr_customer_sk < 10', 'a') ORDER BY nentry DESC, ndoc DESC, word LIMIT 10;
 word | ndoc | nentry
-----+-----+-----
(0 rows)
```

21.10.5 Parsers

Text search parsers are responsible for splitting raw document text into tokens and identifying each token's type, where the set of types is defined by the parser itself. Note that a parser does not modify the text at all — it simply identifies plausible word boundaries. Because of this limited scope, there is less need for application-specific custom parsers than there is for custom dictionaries.

Currently, DWS provides the following built-in parsers: `pg_catalog.default` for English configuration, and `pg_catalog.ngram`, `pg_catalog.zhparser`, and `pg_catalog.pound` for full text search in texts containing Chinese, or both Chinese and English.

The built-in parser is named **`pg_catalog.default`**. It recognizes 23 token types, shown in [Table 21-94](#).

Table 21-94 Default parser's token types

| Alias | Description | Examples |
|------------------------------|--|--|
| <code>asciiword</code> | Word, all ASCII letters | elephant |
| <code>word</code> | Word, all letters | mañana |
| <code>numword</code> | Word, letters and digits | beta1 |
| <code>asciihword</code> | Hyphenated word, all ASCII | up-to-date |
| <code>hword</code> | Hyphenated word, all letters | lógico-matemática |
| <code>numhword</code> | Hyphenated word, letters and digits | postgresql-beta1 |
| <code>hword_asciipart</code> | Hyphenated word part, all ASCII | postgresql in the context
postgresql-beta1 |
| <code>hword_part</code> | Hyphenated word part, all letters | lógico or matemática in the
context lógico-matemática |
| <code>hword_numpart</code> | Hyphenated word part, letters and digits | beta1 in the context
postgresql-beta1 |
| <code>email</code> | Email address | foo@example.com |
| <code>protocol</code> | Protocol head | http:// |
| <code>url</code> | URL | example.com/stuff/index.html |
| <code>host</code> | Host | example.com |
| <code>url_path</code> | URL path | /stuff/index.html, in the
context of a URL |
| <code>file</code> | File or path name | /usr/local/foo.txt, if not
within a URL |
| <code>sfloat</code> | Scientific notation | -1.23E+56 |
| <code>float</code> | Decimal notation | -1.234 |
| <code>int</code> | Signed integer | -1234 |
| <code>uint</code> | Unsigned integer | 1234 |
| <code>version</code> | Version number | 8.3.0 |

| Alias | Description | Examples |
|--------|---------------|--|
| tag | XML tag | |
| entity | XML entity | & |
| blank | Space symbols | (any whitespace or punctuation not otherwise recognized) |

Note: The parser's notion of a "letter" is determined by the database's locale setting, specifically **lc_ctype**. Words containing only the basic ASCII letters are reported as a separate token type, since it is sometimes useful to distinguish them. In most European languages, token types **word** and **asciiword** should be treated alike.

email does not support all valid email characters as defined by RFC 5322. Specifically, the only non-alphanumeric characters supported for email user names are period, dash, and underscore.

It is possible for the parser to identify overlapping tokens in the same piece of text. As an example, a hyphenated word will be reported both as the entire word and as each component:

```
SELECT alias, description, token FROM ts_debug('english','foo-bar-beta1');
  alias | description | token
-----+-----+-----
numhword | Hyphenated word, letters and digits | foo-bar-beta1
hword_asciipart | Hyphenated word part, all ASCII | foo
blank | Space symbols | -
hword_asciipart | Hyphenated word part, all ASCII | bar
blank | Space symbols | -
hword_numpart | Hyphenated word part, letters and digits | beta1
```

This behavior is desirable since it allows searches to work for both the whole compound word and for components. Here is another instructive example:

```
SELECT alias, description, token FROM ts_debug('english','http://example.com/stuff/index.html');
  alias | description | token
-----+-----+-----
protocol | Protocol head | http://
url | URL | example.com/stuff/index.html
host | Host | example.com
url_path | URL path | /stuff/index.html
```

N-gram is a mechanical word segmentation method, and applies to no semantic Chinese segmentation scenarios. The N-gram segmentation method ensures the completeness of the segmentation. However, to cover all the possibilities, it adds unnecessary words to the index, resulting in a large number of index items. N-gram supports Chinese coding, including GBK and UTF-8. Six built-in token types are shown in [Table 21-95](#).

Table 21-95 Token types

| Alias | Description |
|----------|---------------|
| zh_words | chinese words |

| Alias | Description |
|-------------|-----------------|
| en_word | english word |
| numeric | numeric data |
| alnum | alnum string |
| grapsymbol | graphic symbol |
| multisymbol | multiple symbol |

Zhparser is a dictionary-based semantic word segmentation method. The bottom-layer calls the Simple Chinese Word Segmentation (SCWS) algorithm (<https://github.com/hightman/scws>), which applies to Chinese segmentation scenarios. SCWS is a term frequency and dictionary-based mechanical Chinese words engine. It can split a whole paragraph Chinese text into words. The two Chinese coding formats, GBK and UTF-8, are supported. The 26 built-in token types are shown in [Table 21-96](#).

Table 21-96 Token types

| Alias | Description |
|-------|----------------------------|
| A | Adjective |
| B | Differentiation |
| C | Conjunction |
| D | Adverb |
| E | Exclamation |
| F | Position |
| G | Lexeme |
| H | Preceding element |
| I | Idiom |
| J | Acronyms and abbreviations |
| K | Subsequent element |
| L | Common words |
| M | Numeral |
| N | Noun |
| O | Onomatopoeia |
| P | Preposition |
| Q | Quantifiers |

| Alias | Description |
|-------|----------------|
| R | Pronoun |
| S | Space |
| T | Time |
| U | Auxiliary word |
| V | Verb |
| W | Punctuation |
| X | Unknown |
| Y | Interjection |
| Z | Status words |

Pound segments words in a fixed format. It is used to segment to-be-parsed nonsense Chinese and English words that are separated by fixed separators. It supports Chinese encoding (including GBK and UTF8) and English encoding (including ASCII). Pound has six pre-configured token types (as listed in [Table 21-97](#)) and supports five separators (as listed in [Table 21-98](#)). By default, the separator is #. Pound The maximum length of a token is 256 characters.

Table 21-97 Token types

| Alias | Description |
|-------------|-----------------|
| zh_words | chinese words |
| en_word | english word |
| numeric | numeric data |
| alnum | alnum string |
| grapsymbol | graphic symbol |
| multisymbol | multiple symbol |

Table 21-98 Separator types

| Delimiter | Description |
|-----------|-------------------|
| @ | Special character |
| # | Special character |
| \$ | Special character |
| % | Special character |

| Delimiter | Description |
|-----------|-------------------|
| / | Special character |

21.10.6 Dictionaries

21.10.6.1 Overview

A dictionary is used to define stop words, that is, words to be ignored in full-text retrieval.

A dictionary can also be used to normalize words so that different derived forms of the same word will match. A normalized word is called a lexeme.

In addition to improving retrieval quality, normalization and removal of stop words can reduce the size of the **tsvector** representation of a document, thereby improving performance. Normalization and removal of stop words do not always have linguistic meaning. Users can define normalization and removal rules in dictionary definition files based on application environments.

A dictionary is a program that receives a token as input and returns:

- An array of lexemes if the input token is known to the dictionary (note that one token can produce more than one lexeme).
- A single lexeme to replace the original token with a new token to be passed to subsequent dictionaries (a dictionary that does this is called a filtering dictionary).
- An empty array if the input token is known to the dictionary but is a stop word.
- **NULL** if the dictionary does not recognize the token.

DWS provides predefined dictionaries for many languages and also provides five predefined dictionary templates, **Simple**, **Synonym**, **Thesaurus**, **Ispell**, and **Snowball**. These templates can be used to create new dictionaries with custom parameters.

When using full-text retrieval, you are advised to:

- In the text search configuration, configure a parser together with a set of dictionaries to process the parser's output tokens. For each token type that the parser can return, a separate list of dictionaries is specified by the configuration. When a token of that type is found by the parser, each dictionary in the list is consulted in turn, until a dictionary recognizes it as a known word. If it is identified as a stop word, or no dictionary recognizes the token, it will be discarded and not indexed or searched for. Generally, the first dictionary that returns a non-**NULL** output determines the result, and any remaining dictionaries are not consulted. However, a filtering dictionary can replace the input token with a modified one, which is then passed to subsequent dictionaries.
- The general rule for configuring a list of dictionaries is to place first the most narrow, most specific dictionary, then the more general dictionaries, finishing

with a very general dictionary, like a **Snowball** stemmer dictionary or a **Simple** dictionary, which recognizes everything. In the following example, for an astronomy-specific search (**astro_en** configuration), you can configure the token type **asciiword** (ASCII word) with a **Synonym** dictionary of astronomical terms, a general English **Ispell** dictionary, and a **Snowball** English stemmer dictionary:

```
ALTER TEXT SEARCH CONFIGURATION astro_en
ADD MAPPING FOR asciiword WITH astro_syn, english_isspell, english_stem;
```

A filtering dictionary can be placed anywhere in the list, except at the end where it would be useless. Filtering dictionaries are useful to partially normalize words to simplify the task of later dictionaries.

21.10.6.2 Stop Words

Stop words are words that are very common, appear in almost every document, and have no discrimination value. Therefore, they can be ignored in the context of full text searching. Each type of dictionaries treats stop words in different ways. For example, **Ispell** dictionaries first normalize words and then check the list of stop words, while **Snowball** dictionaries first check the list of stop words.

For example, every English text contains words like **a** and **the**, so it is useless to store them in an index. However, stop words affect the positions in **tsvector**, which in turn affect ranking.

```
SELECT to_tsvector('english','in the list of stop words');
to_tsvector
-----
'list':3 'stop':5 'word':6
```

The missing positions 1, 2, and 4 are because of stop words. Ranks calculated for documents with and without stop words are quite different:

```
SELECT ts_rank_cd (to_tsvector('english','in the list of stop words'), to_tsquery('list & stop'));
ts_rank_cd
-----
.05

SELECT ts_rank_cd (to_tsvector('english','list stop words'), to_tsquery('list & stop'));
ts_rank_cd
-----
.1
```

21.10.6.3 Simple Dictionary

A **Simple** dictionary operates by converting the input token to lower case and checking it against a list of stop words. If the token is found in the list, an empty array will be returned, causing the token to be discarded. If it is not found, the lower-cased form of the word is returned as the normalized lexeme. In addition, you can set **Accept** to **false** for **Simple** dictionaries (default: **true**) to report non-stop-words as unrecognized, allowing them to be passed on to the next dictionary in the list.

Precautions

- Most types of dictionaries rely on dictionary configuration files. The name of a configuration file can only be lowercase letters, digits, and underscores (_).
- A dictionary cannot be created in **pg_temp** mode.

- Dictionary configuration files must be stored in UTF-8 encoding. They will be translated to the actual database encoding, if that is different, when they are read into the server.
- Generally, a session will read a dictionary configuration file only once, when it is first used within the session. To modify a configuration file, run the **ALTER TEXT SEARCH DICTIONARY** statement to update and reload the file.

Procedure

Step 1 Create a **Simple** dictionary.

```
CREATE TEXT SEARCH DICTIONARY public.simple_dict (  
    TEMPLATE = pg_catalog.simple,  
    STOPWORDS = english  
);
```

english.stop is the full name of a file of stop words. For details about the syntax and parameters for creating a **Simple** dictionary, see [CREATE TEXT SEARCH DICTIONARY](#).

Step 2 Use the **Simple** dictionary.

```
SELECT ts_lexize('public.simple_dict','Yes');  
ts_lexize  
-----  
{yes}  
(1 row)  
  
SELECT ts_lexize('public.simple_dict','The');  
ts_lexize  
-----  
{}  
(1 row)
```

Step 3 Set **Accept=false** so that the **Simple** dictionary returns **NULL** instead of a lower-cased non-stop word.

```
ALTER TEXT SEARCH DICTIONARY public.simple_dict ( Accept = false );  
SELECT ts_lexize('public.simple_dict','Yes');  
ts_lexize  
-----  
  
(1 row)  
  
SELECT ts_lexize('public.simple_dict','The');  
ts_lexize  
-----  
{}  
(1 row)
```

----End

21.10.6.4 Synonym Dictionary

A synonym dictionary is used to define, identify, and convert synonyms of tokens. Phrases are not supported (use the thesaurus dictionary in [Thesaurus Dictionary](#)).

Examples

- A synonym dictionary can be used to overcome linguistic problems, for example, to prevent an English stemmer dictionary from reducing the word "Paris" to "pari". It is enough to have a **Paris paris** line in the synonym dictionary and put it before the **english_stem** dictionary.

```

SELECT * FROM ts_debug('english', 'Paris');
  alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
asciiword | Word, all ASCII | Paris | {english_stem} | english_stem | {pari}
(1 row)

CREATE TEXT SEARCH DICTIONARY my_synonym (
  TEMPLATE = synonym,
  SYNONYMS = my_synonyms,
  FILEPATH = 'obs://bucket01/obs.example.com accesskey=xxxxx secretkey=xxxxx region=xx-xx-xx'
);

ALTER TEXT SEARCH CONFIGURATION english
  ALTER MAPPING FOR asciiword
    WITH my_synonym, english_stem;

SELECT * FROM ts_debug('english', 'Paris');
  alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

SELECT * FROM ts_debug('english', 'paris');
  alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

ALTER TEXT SEARCH DICTIONARY my_synonym ( CASESENSITIVE=true);

SELECT * FROM ts_debug('english', 'Paris');
  alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {paris}
(1 row)

SELECT * FROM ts_debug('english', 'paris');
  alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
asciiword | Word, all ASCII | Paris | {my_synonym,english_stem} | my_synonym | {pari}
(1 row)

```

The full name of the synonym dictionary file is **my_synonyms.syn**, and the dictionary is stored in 'obs://bucket01/obs.example.com accesskey=xxxxx secretkey=xxxxx region=xx-xx-xx'. For details about the syntax and parameters for creating a synonym dictionary, see [CREATE TEXT SEARCH DICTIONARY](#).

- An asterisk (*) can be placed at the end of a synonym in the configuration file. This indicates that the synonym is a prefix. The asterisk is ignored when the entry is used in `to_tsvector()`, but when it is used in `to_tsquery()`, the result will be a query item with the prefix match marker (see [Manipulating Queries](#)).

Assume that the content in the dictionary file **synonym_sample.syn** is as follows:

```

postgres  pgsql
postgresql  pgsql
postgre pgsql
gogle  googl
indices index*

```

Create and use a dictionary.

```

CREATE TEXT SEARCH DICTIONARY syn (
  TEMPLATE = synonym,
  SYNONYMS = synonym_sample
);

SELECT ts_lexize('syn','indices');

```



```
ts_lexize
-----
{index}
(1 row)

CREATE TEXT SEARCH CONFIGURATION tst (copy=simple);

ALTER TEXT SEARCH CONFIGURATION tst ALTER MAPPING FOR asciiword WITH syn;

SELECT to_tsvector('tst','indices');
to_tsvector
-----
'index':1
(1 row)

SELECT to_tsquery('tst','indices');
to_tsquery
-----
'index':*
(1 row)

SELECT 'indexes are very useful'::tsvector;
tsvector
-----
'are' 'indexes' 'useful' 'very'
(1 row)

SELECT 'indexes are very useful'::tsvector @@ to_tsquery('tst','indices');
?column?
-----
t
(1 row)
```

21.10.6.5 Thesaurus Dictionary

A thesaurus dictionary (sometimes abbreviated as TZ) is a collection of words that include relationships between words and phrases, such as broader terms (BT), narrower terms (NT), preferred terms, non-preferred terms, and related terms. A thesaurus dictionary replaces all non-preferred terms by one preferred term and, optionally, preserves the original terms for indexing as well. A thesaurus dictionary is an extension of the synonym dictionary with added phrase support.

Precautions

- A thesaurus dictionary has the capability to recognize phrases. Therefore, it must remember its state and interact with the parser to check whether it should handle the next token or stop accumulation. The thesaurus dictionary must be configured carefully. For example, if the thesaurus dictionary is assigned to handle only the **asciiword** token, then a thesaurus dictionary definition like **one 7** will not work because token type **uint** is not assigned to the thesaurus dictionary.
- Thesauruses are used during indexing. Any change in the thesaurus dictionary's parameters requires reindexing. For most other dictionary types, small changes such as adding or removing stop words does not force reindexing.

Procedure

Step 1 Create a TZ named **thesaurus_astro**.

thesaurus_astro is a simple astronomical TZ that defines two astronomical word combinations (word+synonym).

```
supernovae stars : sn  
crab nebulae : crab
```

Run the following statement to create the TZ:

```
CREATE TEXT SEARCH DICTIONARY thesaurus_astro (  
  TEMPLATE = thesaurus,  
  DictFile = thesaurus_astro,  
  Dictionary = pg_catalog.english_stem,  
  FILEPATH = 'obs://bucket_name/path accesskey=ak secretkey=sk region=rg'  
);
```

The full name of the thesaurus dictionary file is **thesaurus_astro.ths**, and the dictionary is stored in the **obs://bucket_name/path accesskey=ak secretkey=sk region=rg** directory. **pg_catalog.english_stem** is the subdictionary (a **Snowball** English stemmer) used for input normalization. The subdictionary has its own configuration (for example, stop words), which is not shown here. For details about the syntax and parameters for creating a TZ, see [CREATE TEXT SEARCH DICTIONARY](#).

Step 2 Bind the TZ to the desired token types in the text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION english  
  ALTER MAPPING FOR asciiword, asciihword, hword_asciipart  
  WITH thesaurus_astro, english_stem;
```

Step 3 Use the TZ.

- Test the TZ.

The **ts_lexize** function is not very useful for testing the TZ because the function processes its input as a single token. Instead, you can use the **plainto_tsquery**, **to_tsvector**, or **to_tsquery** function which will break their input strings into multiple tokens.

```
SELECT plainto_tsquery('english','supernova star');  
plainto_tsquery  
-----  
'sn'  
(1 row)  
  
SELECT to_tsvector('english','supernova star');  
to_tsvector  
-----  
'sn':1  
(1 row)  
  
SELECT to_tsquery('english','"supernova star"');  
to_tsquery  
-----  
'sn'  
(1 row)
```

supernova star matches **supernovae stars** in **thesaurus_astro** because the **english_stem** stemmer of the Snowball type is specified in the **thesaurus_astro** definition. The stemmer removed **e** and **s**.

- To index the original phrase, include it in the right-hand part of the definition.

```
supernovae stars : sn supernovae stars  
  
ALTER TEXT SEARCH DICTIONARY thesaurus_astro (  
  DictFile = thesaurus_astro,  
  FILEPATH = 'file:///home/dicts/');  
  
SELECT plainto_tsquery('english','supernova star');  
plainto_tsquery
```

```
-----  
'sn' & 'supernova' & 'star'  
(1 row)
```

----End

21.10.6.6 Ispell Dictionary

The Ispell dictionary template supports morphological dictionaries, which can normalize many different linguistic forms of a word into the same lexeme. For example, an English Ispell dictionary can match all declensions and conjugations of the search term **bank**, such as **banking**, **banked**, **banks**, **banks'**, and **bank's**.

DWS does not provide any predefined Ispell dictionaries or dictionary files. The **.dict** files and **.affix** files support multiple open-source dictionary formats, including **Ispell**, **MySpell**, and **Hunspell**.

Procedure

Step 1 Obtain the dictionary definition file (.dict) and affix file (.affix).

You can use an open-source dictionary. The name extensions of the open-source dictionary may be **.aff** and **.dic**. In this case, you need to change them to **.affix** and **.dict**. In addition, for some dictionary files (for example, Norwegian dictionary files), you need to run the following commands to convert the character encoding to UTF-8:

```
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.affix nn_NO.aff  
iconv -f ISO_8859-1 -t UTF-8 -o nn_no.dict nn_NO.dic
```

Step 2 Create an Ispell dictionary.

```
CREATE TEXT SEARCH DICTIONARY norwegian_ispell (  
  TEMPLATE = ispell,  
  DictFile = nn_no,  
  AffFile = nn_no,  
  FilePath = 'obs://bucket01/obs.example.com accesskey=xxxxx secretkey=xxxxx region=xx-xx-xx'  
);
```

The full name of the Ispell dictionary file is **nn_no.dict** and **nn_no.affix**, and the dictionary is stored in 'obs://bucket01/obs.example.com accesskey=xxxxx secretkey=xxxxx region=xx-xx-xx'. For details about the syntax and parameters for creating an Ispell dictionary, see [CREATE TEXT SEARCH DICTIONARY](#).

Step 3 Use the Ispell dictionary to split compound words.

```
SELECT ts_lexize('norwegian_ispell', 'sjokoladefabrikk');  
      ts_lexize  
-----  
{sjokolade,fabrikk}  
(1 row)
```

MySpell does not support compound words. **Hunspell** supports compound words. DWS supports only the basic compound word operations of **Hunspell**. Generally, an Ispell dictionary recognizes a limited set of words, so they should be followed by another broader dictionary, for example, a Snowball dictionary, which recognizes everything.

----End

21.10.6.7 Snowball Dictionary

A **Snowball** dictionary is based on a project by Martin Porter and is used for stem analysis, providing stemming algorithms for many languages. DWS provides predefined **Snowball** dictionaries of many languages. You can query the **PG_TS_DICT** system catalog to view the predefined **Snowball** dictionaries and supported stemming algorithms.

Snowball dictionaries identify all inputs as recognized, whether simplified or not, so they should be placed at the end of the dictionary list. It is useless to place it before any other dictionary because a token will never pass it through to the next dictionary.

For details about the syntax of **Snowball** dictionaries, see [CREATE TEXT SEARCH DICTIONARY](#).

21.10.7 Configuration Examples

Text search configuration specifies the following components required for converting a document into a **tsvector**:

- A parser, decomposes a text into tokens.
- Dictionary list, converts each token into a lexeme.

Each time when the **to_tsvector** or **to_tsquery** function is invoked, a text search configuration is required to specify a processing procedure. The GUC parameter **default_text_search_config** specifies the default text search configuration, which will be used if the text search function does not explicitly specify a text search configuration.

DWS provides some predefined text search configurations. You can also create user-defined text search configurations. In addition, to facilitate the management of text search objects, multiple **gsql** meta-commands are provided to display related information. For details, see "Meta-Command Reference" in the Data Warehouse Service (DWS) *Tool Guide*.

Procedure

- Step 1** Create a text search configuration **ts_conf** by copying the predefined text search configuration **english**.

```
CREATE TEXT SEARCH CONFIGURATION ts_conf ( COPY = pg_catalog.english );  
CREATE TEXT SEARCH CONFIGURATION
```

- Step 2** Create a **Synonym** dictionary.

Assume that the definition file **pg_dict.syn** of the **Synonym** dictionary contains the following contents:

```
postgres pg  
pgsql pg  
postgresql pg
```

Run the following statement to create the **Synonym** dictionary:

```
CREATE TEXT SEARCH DICTIONARY pg_dict (  
    TEMPLATE = synonym,  
    SYNONYMS = pg_dict,  
    FILEPATH = 'obs://bucket01/obs.example.com accesskey=xxxxx secretkey=xxxxx region=xx-xx-xx'  
);
```

- Step 3** Create an **Ispell** dictionary **english_ispell** (the dictionary definition file is from the open source dictionary).

```
CREATE TEXT SEARCH DICTIONARY english_ispell (  
    TEMPLATE = ispell,  
    DictFile = english,  
    AffFile = english,  
    StopWords = english,  
    FILEPATH = 'obs://bucket01/obs.example.com accesskey=xxxxx secretkey=xxxxx region=xx-xx-xx'  
);
```

- Step 4** Modify the text search configuration **ts_conf** and change the dictionary list for tokens of certain types. For details about token types, see [Parsers](#).

```
ALTER TEXT SEARCH CONFIGURATION ts_conf  
    ALTER MAPPING FOR asciiword, asciihword, hword_asciipart,  
        word, hword, hword_part  
    WITH pg_dict, english_ispell, english_stem;
```

- Step 5** In the text search configuration, set non-index or set the search for tokens of certain types.

```
ALTER TEXT SEARCH CONFIGURATION ts_conf  
    DROP MAPPING FOR email, url, url_path, sfloat, float;
```

- Step 6** Use the text retrieval commissioning function **ts_debug()** to test the text search configuration **ts_conf**.

```
SELECT * FROM ts_debug('ts_conf', '  
PostgreSQL, the highly scalable, SQL compliant, open source object-relational  
database management system, is now undergoing beta testing of the next  
version of our software.  
' );
```

- Step 7** You can set the default text search configuration of the current session to **ts_conf**. This setting is valid only for the current session.

```
\dF+ ts_conf  
    Text search configuration "public.ts_conf"  
Parser: "pg_catalog.default"  
    Token      |      Dictionaries  
-----+-----  
asciihword    | pg_dict,english_ispell,english_stem  
asciiword     | pg_dict,english_ispell,english_stem  
file          | simple  
host          | simple  
hword         | pg_dict,english_ispell,english_stem  
hword_asciipart | pg_dict,english_ispell,english_stem  
hword_numpart | simple  
hword_part    | pg_dict,english_ispell,english_stem  
int           | simple  
numhword      | simple  
numword       | simple  
uint          | simple  
version       | simple  
word          | pg_dict,english_ispell,english_stem  
  
SET default_text_search_config = 'public.ts_conf';  
SET  
SHOW default_text_search_config;  
default_text_search_config  
-----  
public.ts_conf  
(1 row)
```

----End

21.10.8 Testing and Debugging Text Search

21.10.8.1 Testing a Configuration

The function `ts_debug` allows easy testing of a text search configuration.

```
ts_debug([ config regconfig, ] document text,
        OUT alias text,
        OUT description text,
        OUT token text,
        OUT dictionaries regdictionary[],
        OUT dictionary regdictionary,
        OUT lexemes text[])
returns setof record
```

ts_debug displays information about every token of document as produced by the parser and processed by the configured dictionaries. It uses the configuration specified by **config**, or **default_text_search_config** if that argument is omitted.

ts_debug returns one row for each token identified in the text by the parser. The columns returned are:

- alias text — short name of the token type
- description text — description of the token type
- token text — text of the token
- **dictionaries regdictionary[]** — the dictionaries selected by the configuration for this token type
- **dictionary regdictionary**: the dictionary that recognized the token, or NULL if none did
- **lexemes text[]**: the lexeme(s) produced by the dictionary that recognized the token, or NULL if none did; an empty array ({}) means the token was recognized as a stop word

Here is a simple example:

```
SELECT * FROM ts_debug('english','a fat cat sat on a mat - it ate a fat rats');
 alias | description | token | dictionaries | dictionary | lexemes
-----+-----+-----+-----+-----+-----
asciiword | Word, all ASCII | a | {english_stem} | english_stem | {}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | fat | {english_stem} | english_stem | {fat}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | cat | {english_stem} | english_stem | {cat}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | sat | {english_stem} | english_stem | {sat}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | on | {english_stem} | english_stem | {}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | a | {english_stem} | english_stem | {}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | mat | {english_stem} | english_stem | {mat}
blank | Space symbols | | {} | | 
blank | Space symbols | - | {} | | 
asciiword | Word, all ASCII | it | {english_stem} | english_stem | {}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | ate | {english_stem} | english_stem | {ate}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | a | {english_stem} | english_stem | {}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | fat | {english_stem} | english_stem | {fat}
blank | Space symbols | | {} | | 
asciiword | Word, all ASCII | rats | {english_stem} | english_stem | {rat}
(24 rows)
```

21.10.8.2 Testing a Parser

The **ts_parse** function allows direct testing of a text search parser.

```
ts_parse(parser_name text, document text,  
OUT tokid integer, OUT token text) returns setof record
```

ts_parse parses the given **document** and returns a series of records, one for each token produced by parsing. Each record includes a **tokid** showing the assigned token type and a **token** which is the text of the token. For example:

```
SELECT * FROM ts_parse('default', '123 - a number');  
tokid | token  
-----+-----  
22 | 123  
12 |  
12 | -  
1 | a  
12 |  
1 | number  
(6 rows)  
ts_token_type(parser_name text, OUT tokid integer,  
OUT alias text, OUT description text) returns setof record
```

ts_token_type returns a table which describes each type of token the specified parser can recognize. For each token type, the table gives the integer **tokid** that the parser uses to label a token of that type, the **alias** that names the token type in configuration commands, and a short description. For example:

```
SELECT * FROM ts_token_type('default');  
tokid | alias | description  
-----+-----  
1 | asciiword | Word, all ASCII  
2 | word | Word, all letters  
3 | numword | Word, letters and digits  
4 | email | Email address  
5 | url | URL  
6 | host | Host  
7 | sfloat | Scientific notation  
8 | version | Version number  
9 | hword_numpart | Hyphenated word part, letters and digits  
10 | hword_part | Hyphenated word part, all letters  
11 | hword_asciipart | Hyphenated word part, all ASCII  
12 | blank | Space symbols  
13 | tag | XML tag  
14 | protocol | Protocol head  
15 | numhword | Hyphenated word, letters and digits  
16 | asciihword | Hyphenated word, all ASCII  
17 | hword | Hyphenated word, all letters  
18 | url_path | URL path  
19 | file | File or path name  
20 | float | Decimal notation  
21 | int | Signed integer  
22 | uint | Unsigned integer  
23 | entity | XML entity  
(23 rows)
```

21.10.8.3 Testing a Dictionary

The **ts_lexize** function facilitates dictionary testing.

ts_lexize(dict regdictionary, token text) returns text[]: **ts_lexize** returns an array of lexemes if the input **token** is known to the dictionary, or an empty array if the token is known to the dictionary but it is a stop word, or **NULL** if it is an unknown word.

For example:

```
SELECT ts_lexize('english_stem', 'stars');
ts_lexize
-----
{star}

SELECT ts_lexize('english_stem', 'a');
ts_lexize
-----
{}
```

NOTICE

The **ts_lexize** function expects a single **token**, not text.

21.10.9 Limitations

The current limitations of DWS's full text search are:

- The length of each lexeme must be less than 2 KB.
- The length of a **tsvector** (lexemes + positions) must be less than 1 megabyte.
- Position values in **tsvector** must be greater than 0 and no more than 16,383.
- No more than 256 positions per lexeme. Excessive positions, if any, will be discarded.
- The number of nodes (lexemes + operators) in a tsquery must be less than 32,768.

21.11 System Operation

DWS runs SQL statements to perform different system operations, such as setting variables, displaying the execution plan, and collecting garbage data.

Setting Variables

For details about how to set various parameters for a session or transaction, see [SET](#).

Displaying the Execution Plan

For details about how to display the execution plan that DWS makes for SQL statements, see [EXPLAIN](#).

Specifying a Checkpoint in Transaction Logs

By default, WALs periodically specify checkpoints in a transaction log. **CHECKPOINT** forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system. For details, see [CHECKPOINT](#).

Collecting Unnecessary Data

For details about how to collect garbage data and analyze a database as required,
For details, see [VACUUM](#).

Collecting statistics

For details about how to collect statistics on tables in databases, For details, see
[ANALYZE](#) | [ANALYSE](#).

Setting the Constraint Check Mode for the Current Transaction

For details about how to set the constraint check mode for the current transaction,
For details, see [SET CONSTRAINTS](#).

21.12 Transaction Management

DWS supports the ACID properties of database transactions. It provides the **READ COMMITTED** and **REPEATABLE READ** isolation levels of transactions.



WARNING

- In scenarios where the client explicitly starts a transaction or manually disables "autocommit", you must manually execute a commit command to commit the transaction.
 - You are advised to optimize the statements that take longer than 30 minutes to execute.
 - Avoid business operations that take longer than 2 hours to execute to prevent issues such as long transactions and long locks.
 - For more information about development design proposal, see "DWS Development Design Proposal" in the *Data Warehouse Service Developer Guide*.
-

Concepts

- A transaction refers to an operation that consists of multiple steps, either all successful or all failed.
- A database transaction is a logical unit in the execution process of DBMS. It consists of a limited number of database operations in a certain sequence (generally all the operations between **BEGIN TRANSACTION** and **END TRANSACTION**). These operations are either fully completed or not performed at all.

Purposes

The purposes of database transactions are as follows:

- To provide a method for restoring the database operation sequence from a failure while keeping database consistency even in case of system failure.

- To provide isolation between programs accessing a database concurrently, so that they do not interfere one another in this process.

Transaction Execution Process

After a transaction is committed in DBMS, DBMS needs to ensure that all operations in the transaction are successfully completed and the results are permanently stored in the database. If an operation in the transaction fails, all the operations in the transaction must be rolled back to the status before the transaction is executed. Transactions run independently and do not interfere with each other or affect database running.

Transaction Properties

A transaction has atomicity, consistency, isolation, and durability (ACID) properties.

- **Atomicity:** All the operations in a transaction are inseparable. They are either fully completed or not executed at all.
For example, if account A transfers an amount of money to account B, \$500 are deducted from account A and \$500 added to account B. If the amount fails to be added to account B, it cannot be deducted from account A. If atomicity is not guaranteed, the account balances will be inconsistent.
- **Consistency:** A transaction can only bring the database from one valid state to another, maintaining database invariants. Any data written to the database must be valid according to all defined rules, including data accuracy, concatenation, and spontaneous execution of scheduled tasks.
For example, if account A transfers \$500 to account B, \$500 is deducted from account A and added to account B. The sum of the deducted amount (-\$500) and the added amount (+500) should be 0. The total account balance of accounts A and B remains unchanged, no matter whether the money is transferred.
- **Isolation:** The execution of a transaction cannot be interfered by other transactions. Isolation means that the operations inside a transaction and data used are isolated from other concurrent transactions. The concurrent transactions do not affect each other.
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**.
- **Durability:** Once a transaction is committed, its modifications are permanently saved to the database. Even if the system is faulty, committed modifications will not be lost.

Table 21-99 ACID usage

| ACID | Purpose |
|-----------|--|
| Atomicity | Concurrency control and fault recovery |

| ACID | Purpose |
|-------------|---|
| Consistency | SQL integrity constraints (primary key and foreign key constraints) |
| Isolation | Concurrency control |
| Durability | Fault recovery |

Common concurrency control technologies include lock-based and timestamp-based concurrency control. DWS uses the two-phase lock technology for DDL statements and uses multi-version concurrency control (MVCC) for DML statements. DWS databases fault recovery is based on WAL logs. MVCC mainly uses redo logs to ensure transaction read/write consistency.

Redo logs are critical components in database systems that ensure data durability and enable fault recovery. These logs record the modification operations—not the data before or after—performed by transactions. In the event of a failure, such as a system breakdown or crash, the database can replay these operations to restore data to a consistent state before the fault occurred.

Redo log recovery process: After a crash and subsequent restart, the database performs recovery using the redo log, which is known as Write-Ahead Log (WAL). It scans log entries from the last checkpoint—representing committed transactions that were not yet written to disk—and re-executes these operations to restore lost in-memory changes. Simultaneously, uncommitted transactions are rolled back using undo operations to reverse partial modifications. This ensures data is restored to a consistent and complete state, preserving durability of committed transactions and atomicity of incomplete ones. This mechanism is the core basis for databases to enable high reliability.

Isolation Levels

Isolation prevents data inconsistency during the execution of concurrent transactions. A transaction isolation level specifies how concurrent transactions process the same object.

In DWS, transaction isolation levels are controlled by the GUC parameter **transaction_isolation** or the **SET TRANSACTION** syntax. The following isolation levels are supported. The default isolation level is **READ COMMITTED**.

- **READ COMMITTED:** Only committed data is read.
- **READ UNCOMMITTED:** DWS does not support **READ UNCOMMITTED**. If **READ UNCOMMITTED** is set, **READ COMMITTED** is used instead.
- **REPEATABLE READ:** Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.
- **SERIALIZABLE:** DWS does not support **SERIALIZABLE**. If **SERIALIZABLE** is set, **REPEATABLE READ** is used instead.

Transaction Control Syntax

- Starting a transaction

DWS starts a transaction using **START TRANSACTION** and **BEGIN**. For details, see [START TRANSACTION](#) and [BEGIN](#).

- Setting a transaction

DWS sets a transaction using **SET TRANSACTION** or **SET LOCAL TRANSACTION**. For details, see [SET TRANSACTION](#).

- Committing a transaction

DWS commits all operations of a transaction using **COMMIT** or **END**. For details, see [COMMIT | END](#).

- Rolling back a transaction

If a fault occurs during a transaction and the transaction cannot proceed, the system performs rollback to cancel all the completed database operations related to the transaction. For details, see [ROLLBACK](#).

NOTE

If an execution request (not in a transaction block) received in the database contains multiple statements, the statements will be packed into a transaction. If one of the statements fails, the entire request will be rolled back.

- Other operations on transactions

- **SAVEPOINT** establishes a new savepoint within the current transaction. The transaction can be rolled back to the savepoint. You can roll back the commands executed after a savepoint but retain the commands executed before the savepoint. For details, see [SAVEPOINT](#).
- **ROLLBACK TO SAVEPOINT** rolls back a transaction to a savepoint. It implicitly deletes all the savepoints established after that savepoint. For details, see [ROLLBACK TO SAVEPOINT](#).
- **RELEASE SAVEPOINT** deletes a savepoint in a transaction. For details, see [RELEASE SAVEPOINT](#).

Transaction Example

A customer buys a \$100 item in a store using an e-payment account. At least two operations are involved: 1. \$100 is deducted from the customer's account. 2. \$100 is added to the store's account. In DBMS, the two operations must be both completed or not executed at all.

- Create sample data.

Create an account balance table and insert data. (Assume the store's and the customer's accounts each have \$500.)

```
CREATE TABLE customer_info (  
  NAME VARCHAR(32) PRIMARY KEY,  
  MONEY INTEGER  
);  
INSERT INTO customer_info (name, money) VALUES ('buyer', 500), ('shop', 500);
```

The table data shows that the store and customer each have \$500.

```
SELECT * FROM customer_info;
```

| name | money |
|----------|-------|
| buyer | 500 |
| shop | 500 |
| (2 rows) | |

2. Simulate a successful transaction.

Deduct \$100 from the customer's account and add \$100 to the store's account.

```
UPDATE customer_info SET money = money-100 WHERE name IN (SELECT name FROM
customer_info WHERE name = 'buyer');
UPDATE customer_info SET money = money+100 WHERE name IN (SELECT name FROM
customer_info WHERE name = 'shop');

SELECT * FROM customer_info;
```

| name | money |
|----------|-------|
| buyer | 400 |
| shop | 600 |
| (2 rows) | |

3. Restore initial values.

```
UPDATE customer_info SET money=500;
select * from customer_info;
```

| name | money |
|----------|-------|
| shop | 500 |
| buyer | 500 |
| (2 rows) | |

4. Simulate a transaction failure.

\$100 is deducted from the customer's account but fails to be added to the store's account.

a. Deduct \$100 from the customer's account.

```
UPDATE customer_info SET money = money-100 WHERE name IN (SELECT name FROM
customer_info WHERE name = 'buyer');
```

b. The store finds a payment problem and terminates subsequent operations. An error is reported when the amount of money is added to the store's account. The execution of the following statement is terminated. (Only the store thinks that there is a problem with payment.)

```
UPDATE customer_info SET money = money+100 WHERE name IN (SELECT name FROM
customer_info WHERE name = 'shop');
```

c. Query the account balances. The consumer has paid \$100 but store does not receive it.

```
SELECT * FROM customer_info;
```

| name | money |
|----------|-------|
| buyer | 400 |
| shop | 500 |
| (2 rows) | |

Without ACID properties, the account balances will be incorrect once an error occurs during SQL statement execution..

Simulate the rollback of an abnormal database transaction.

1. Restore initial values.

```
UPDATE customer_info SET money=500;
```

2. Deduct \$100 from the customer's account.

```
BEGIN TRANSACTION;  
UPDATE customer_info SET money = money-100 WHERE name IN (SELECT name FROM  
customer_info WHERE name = 'buyer');
```

3. An error is reported when the amount of money is added to the store's account. The execution of the following statement is terminated.

```
UPDATE customer_info SET money = money+100 WHERE name IN (SELECT name FROM  
customer_info WHERE name = 'shop');
```

4. Roll back the transaction. All the completed database operations related to the transaction are canceled.

```
ERROR: syntax error at or near "shop"  
LINE 1: ...e IN (SELECT name FROM customer_info WHERE name = ''shop'');  
END TRANSACTION;  
ROLLBACK
```

5. Query the account balances. The query result shows that the account balances remain unchanged. If an error occurs during transaction execution, the database is rolled back to the state before the transaction starts. The integrity of the database is not damaged.

```
SELECT * FROM customer_info;
```

| name | money |
|----------|-------|
| buyer | 500 |
| shop | 500 |
| (2 rows) | |

Two-Phase Transaction

DWS uses the distributed shared nothing architecture. Table data is distributed on different nodes. One or more statements on the client may modify data on multiple nodes at the same time. In this case, a distributed transaction is generated. DWS uses two-phase commit transactions to ensure data consistency and atomicity in distributed transactions. Two-phase commit divides transaction commit into two phases, usually for transactions that contain write operations. When data is written to different nodes, the atomicity requirement of the transaction must be met, that is, either all data is committed or all data is rolled back.

Two-phase commit is not supported in the following scenarios:

- Explicit two-phase commit of **PREPARE TRANSACTION** is not supported.

```
BEGIN;  
PREPARE TRANSACTION 'p1';
```

```
ERROR: Explicit prepare transaction is not supported.
```

- The file mappings of system catalogs cannot be modified in a two-phase transaction.

```
REINDEX TABLE pg_class;
```

```
ERROR: cannot PREPARE a transaction that modified relation mapping.
```

- Transaction snapshots cannot be committed and exported in cross-node transactions.

```
BEGIN;  
CREATE TABLE t1(a int);  
SELECT pg_export_snapshot();  
END;
```

```
ERROR: cannot PREPARE a transaction that has exported snapshots.
```

21.13 DDL Syntax

21.13.1 DDL Syntax Overview

Data definition language (DDL) is used to define or modify an object in a database, such as a table, index, or view.

NOTE

DWS does not support DDL if its CN is unavailable. For example, if a CN in the cluster is faulty, creating a database or table will fail.

Defining a Database

A database is the warehouse for organizing, storing, and managing data. Defining a database includes: creating a database, altering the database attributes, and dropping the database. The following table lists the related SQL statements.

Table 21-100 SQL statements for defining a database

| Function | SQL Statement |
|---------------------------|------------------------|
| Create a database | CREATE DATABASE |
| Alter database attributes | ALTER DATABASE |
| Delete a database | DROP DATABASE |

Defining a Schema

A schema is the set of a group of database objects and is used to control the access to the database objects. The following table lists the related SQL statements.

Table 21-101 SQL statements for defining a schema

| Function | SQL Statement |
|-------------------------|----------------------|
| Create a schema | CREATE SCHEMA |
| Alter schema attributes | ALTER SCHEMA |
| Delete a schema | DROP SCHEMA |

Defining a Table

A table is a special data structure in a database and is used to store data objects and the relationship between data objects. The following table lists the related SQL statements.

Table 21-102 SQL statements for defining a table

| Function | SQL Statement |
|----------------------------------|---------------------|
| Create a table | CREATE TABLE |
| Alter table attributes | ALTER TABLE |
| Alter a table name | RENAME TABLE |
| Delete a table | DROP TABLE |
| Delete all the data from a table | TRUNCATE |

Defining a Partitioned Table

A partitioned table is a special data structure in a database and is used to store data objects and the relationship between data objects. The following table lists the related SQL statements.

Table 21-103 SQL statements for defining a partitioned table

| Function | SQL Statement |
|------------------------------------|-------------------------------|
| Create a partitioned table | CREATE TABLE PARTITION |
| Editing a partition | ALTER TABLE PARTITION |
| Alter partitioned table attributes | ALTER TABLE PARTITION |
| Delete a partition | ALTER TABLE PARTITION |
| Delete a partitioned table | DROP TABLE |

Defining an Index

An index indicates the sequence of values in one or more columns in the database table. The database index is a data structure that improves the speed of data access to specific information in a database table. The following table lists the related SQL statements.

Table 21-104 SQL statements for defining an index

| Function | SQL Statement |
|------------------------|---------------------|
| Create an index | CREATE INDEX |
| Alter index attributes | ALTER INDEX |
| Delete an index | DROP INDEX |
| Rebuild an index | REINDEX |

Defining a Role

A role is used to manage rights. For database security, all management and operation rights can be assigned to different roles. The following table lists the related SQL statements.

Table 21-105 SQL statements for defining a role

| Function | SQL Statement |
|-----------------------|--------------------|
| Create a role | CREATE ROLE |
| Alter role attributes | ALTER ROLE |
| Delete a role | DROP ROLE |

Defining a User

A user is used to log in to a database. Different rights can be assigned to users for managing data accesses and operations of users. The following table lists the related SQL statements.

Table 21-106 SQL statements for defining a user

| Function | SQL Statement |
|-----------------------|--------------------|
| Create a user | CREATE USER |
| Alter user attributes | ALTER USER |
| Delete a user | DROP USER |

Defining a Redaction Policy

Data redaction is to protect sensitive data by masking or changing data. You can create a data redaction policy for a specific table object and specify the effective scope of the policy. You can also add, modify, and delete redaction columns. The following table lists the related SQL statements.

Table 21-107 SQL statements for managing redaction policies

| Function | SQL Statement |
|---|--------------------------------|
| Create a data redaction policy | CREATE REDACTION POLICY |
| Modify a data redaction policy applied to a specified table | ALTER REDACTION POLICY |
| Delete a data redaction policy applied to a specified table | DROP REDACTION POLICY |

Defining Row-Level Access Control

Row-level access control policies control the visibility of rows in database tables. In this way, the same SQL query may return different results for different users. The following table lists the related SQL statements.

Table 21-108 SQL statements for row-level access control

| Function | SQL Statement |
|---|---|
| Create a row-level access control policy | CREATE ROW LEVEL SECURITY POLICY |
| Modify an existing row-level access control policy | ALTER ROW LEVEL SECURITY POLICY |
| Delete a row-level access control policy from a table | DROP ROW LEVEL SECURITY POLICY |

Defining a Stored Procedure

A stored procedure is a set of SQL statements for achieving specific functions and is stored in the database after compiling. Users can specify a name and provide parameters (if necessary) to execute the stored procedure. The following table lists the related SQL statements.

Table 21-109 SQL statements for defining a stored procedure

| Function | SQL Statement |
|---------------------------|-------------------------|
| Create a stored procedure | CREATE PROCEDURE |

| Function | SQL Statement |
|---------------------------|-----------------------|
| Delete a stored procedure | DROP PROCEDURE |

Defining a Function

In DWS, a function is similar to a stored procedure, which is a set of SQL statements. The function and stored procedure are used the same. The following table lists the related SQL statements.

Table 21-110 SQL statements for defining a function

| Function | SQL Statement |
|---------------------------|------------------------|
| Create a function | CREATE FUNCTION |
| Alter function attributes | ALTER FUNCTION |
| Delete a function | DROP FUNCTION |

Defining a View

A view is a virtual table exported from one or several basic tables. The view is used to control data accesses for users. The following table lists the related SQL statements.

Table 21-111 SQL statements for defining a view

| Function | SQL Statement |
|---------------|--------------------|
| Create a view | CREATE VIEW |
| Delete a view | DROP VIEW |

Defining a Cursor

To process SQL statements, the stored procedure process assigns a memory segment to store context association. Cursors are handles or pointers to context regions. With a cursor, the stored procedure can control alterations in context areas.

Table 21-112 SQL statements for defining a cursor

| Function | SQL Statement |
|-----------------|---------------|
| Create a cursor | CURSOR |
| Move a cursor | MOVE |

| Function | SQL Statement |
|----------------------------|---------------|
| Extract data from a cursor | FETCH |
| Close a cursor | CLOSE |

Altering or Ending a Session

A session is a connection established between the user and the database. The following table lists the related SQL statements.

Table 21-113 SQL statements related to sessions

| Function | SQL Statement |
|-----------------|----------------------------------|
| Alter a session | ALTER SESSION |
| End a session | ALTER SYSTEM KILL SESSION |

Defining a Resource Pool

A resource pool is a system catalog used by the resource load management module to specify attributes related to resource management, such as Cgroups. The following table lists the related SQL statements.

Table 21-114 SQL statements for defining a resource pool

| Function | SQL Statement |
|----------------------------|-----------------------------|
| Create a resource pool | CREATE RESOURCE POOL |
| Change resource attributes | ALTER RESOURCE POOL |
| Delete a resource pool | DROP RESOURCE POOL |

Defining Synonyms

A synonym is a special database object compatible with Oracle. It is used to store the mapping between a database object and another. Currently, only synonyms can be used to associate the following database objects: tables, views, functions, and stored procedures. The following table lists the related SQL statements.

Table 21-115 SQL statements for managing synonyms

| Function | SQL Statement |
|---------------------|-----------------------|
| Creating a synonym | CREATE SYNONYM |
| Modifying a synonym | ALTER SYNONYM |

| Function | SQL Statement |
|--------------------|---------------------|
| Deleting a synonym | DROP SYNONYM |

Defining Text Search Configuration

A text search configuration specifies a text search parser that can divide a string into tokens, plus dictionaries that can be used to determine which tokens are of interest for searching. The following table lists the related SQL statements.

Table 21-116 SQL statements for configuring text search

| Function | SQL Statement |
|------------------------------------|---|
| Create a text search configuration | CREATE TEXT SEARCH CONFIGURATION |
| Modify a text search configuration | ALTER TEXT SEARCH CONFIGURATION |
| Delete a text search configuration | DROP TEXT SEARCH CONFIGURATION |

Defining a Full-text Retrieval Dictionary

A dictionary is used to identify and process specific words during full-text retrieval. Dictionaries are created by using predefined templates (defined in the **PG_TS_TEMPLATE** system catalog). Five types of dictionaries can be created, **Simple**, **Ispell**, **Synonym**, **Thesaurus**, and **Snowball**. Each type of dictionaries is used to handle different tasks. The following table lists the related SQL statements.

Table 21-117 SQL statements for a full-text search dictionary

| Function | SQL Statement |
|---|--------------------------------------|
| Create a full-text retrieval dictionary | CREATE TEXT SEARCH DICTIONARY |
| Modify a full-text retrieval dictionary | ALTER TEXT SEARCH DICTIONARY |
| Delete a full-text retrieval dictionary | DROP TEXT SEARCH DICTIONARY |

Defining Query Filtering Rules

You can create query filter rules to block slow SQL statements identified in advance. The following table lists the related SQL statements.

Table 21-118 SQL statements for querying filtering rules

| Function | SQL Statement |
|----------------------------------|--------------------------|
| Creating a Query Filtering Rule | CREATE BLOCK RULE |
| Modifying a Query Filtering Rule | ALTER BLOCK RULE |
| Deleting a Query Filtering Rule | DROP BLOCK RULE |

21.13.2 Databases, Tablespaces, Schemas, and Data Types

21.13.2.1 CREATE DATABASE

Function

CREATE DATABASE creates a database.

DWS creates a database by copying a database template. DWS has two default template databases **template0** and **template1** and a default user database **gaussdb**. By default, a new database is created based on **template1**. The character encoding of **template1** cannot be changed. If you want to change it, you can set the **template** parameter to **template0**.

Precautions

- A user that has the **CREATEDB** permission or a sysadmin can create a database.
- **CREATE DATABASE** cannot be executed within a transaction block.
- Errors along the line of "could not initialize database directory" are most likely related to insufficient permissions on the data directory, a full disk, or other file system problems.

Syntax

```
CREATE DATABASE database_name
[ [ WITH ] { [ OWNER [=] user_name ] |
  [ TEMPLATE [=] template ] |
  [ ENCODING [=] encoding ] |
  [ LC_COLLATE [=] lc_collate ] |
  [ LC_CTYPE [=] lc_ctype ] |
  [ DBCOMPATIBILITY [=] compatibility_type ] |
  [ DBCOMPATIBILITY_BEHAVIOR [=] opt_compat_behavior ] |
  [ CONNECTION LIMIT [=] connlimit ] }[...];
```

Parameter Description

Table 21-119 CREATE DATABASE parameters

| Parameter | Description | Value Range |
|---------------|--|---|
| database_name | Name of the database to be created | A string, which must comply with the naming convention . |
| user_name | Specifies the owner of the new database. By default, the owner of the database is the current user. | An existing user name. |
| template | Specifies the name of the template used to create a database. If it is not specified, the system uses template1 by default. | Value range:
template0 or template1 . <ul style="list-style-type: none">• template0: The character set encoding can be specified if template0 is used.• template1 (default value): The character set encoding cannot be specified if template1 is used. |
| encoding | Specifies the character set encoding used by the new database. The value can be a string (for example, SQL_ASCII) or an integer. By default, the encoding format of the template database is used.

The encoding formats of the template databases template0 and template1 vary based on OS environments by default. | Common values: GBK , UTF8 , Latin1 , and SQL_ASCII . |

| Parameter | Description | Value Range |
|---------------------|--|---|
| lc_collate | <p>Specifies the collation order to use in the new database. For example, this parameter can be set using <code>lc_collate = 'zh_CN.gbk'</code>.</p> <p>The use of this parameter affects the sort order applied to strings, for example, in queries with ORDER BY, as well as the order used in indexes on text columns. The default is to use the collation order of the template database. For other restrictions, see Character Set Encoding Description.</p> | Valid collation order |
| lc_ctype | <p>Specifies the character classification to use in the new database. For example, this parameter can be set using <code>lc_ctype = 'zh_CN.gbk'</code>.</p> <p>The use of this parameter affects the categorization of characters, for example, lower, upper and digit. The default is to use the character classification of the template database.</p> | Valid character classification |
| compatibility_type | Specifies the compatible database type. If this parameter is not specified, the default value ORA is used. | The value can be ORA , TD , or MySQL , indicating Oracle, Teradata, or MySQL database, respectively. |
| opt_compat_behavior | <p>Specifies the compatibility behavior of the database. This parameter is supported only by clusters of version 9.1.0 or later.</p> <p>If this parameter is not specified, the default value NO_BEHAVIOR is used, indicating no special behavior.</p> | <p>The value can be td_rtrim or pg_char.</p> <ul style="list-style-type: none">• td_rtrim: removes the trailing part of a variable-length character string in Teradata compatibility mode.• pg_char: converts the varchar type to the nvarchar2 type in PostgreSQL compatibility mode. |

| Parameter | Description | Value Range |
|-----------|---|--|
| conlimit | <p>Number of concurrent connections to a database. The default value -1 means no limit.</p> <ul style="list-style-type: none">• This limit does not apply to system administrators.• To ensure that a cluster runs properly, the minimum value of CONNECTION LIMIT is the number of CNs in the cluster. because when a cluster runs ANALYZE on a CN, other CNs will connect to the running CN for metadata synchronization. For example, if there are three CNs in the cluster, set CONNECTION LIMIT to 3 or a larger value. | An integer greater than or equal to -1 |

Character Set Encoding Description

- To view the character encoding of the current database, run the **show server_encoding;** command.
- To make your database compatible with most characters, you are advised to use the UTF-8 encoding when creating a database.
- The character set encoding of the new database must be compatible with the language settings (**LC_COLLATE** and **LC_CTYPE**).
- If the language is **C** (or equivalently **POSIX**), then all encoding modes are allowed, but for other language settings only the encoding mode for character sets consistent with that of the locale will work properly.
- The encoding and locale settings must match those of the template database, except when template0 is used as template. This is because other databases might contain data that does not match the specified encoding, or might contain indexes whose sort ordering is affected by **LC_COLLATE** and **LC_CTYPE**. Copying such data would result in a database that is corrupt according to the new settings. template0, however, is known to not contain any data or indexes that would be affected.
- Supported encoding depends on the environment. If the message "invalid locale name" is displayed, run the **locale -a** command to check the encoding set supported by the environment.
- When the specified character encoding set is **GBK**, some uncommon Chinese characters cannot be directly used as object names. This is because when the encoding range of the second byte of GBK is between 0x40 and 0x7E, the byte encoding overlaps with the ASCII character @A-Z[]^_`a-z{ }. @[]^_{ } is an operator in the database. If it is directly used as an object name, a syntax error will be reported. For example, the GBK hexadecimal code is **0x8240**, and the second byte is **0x40**, which is the same as the ASCII character @. Therefore, the character cannot be used as an object name. If you really want

to use it, you can avoid this problem by adding double quotation marks when creating and accessing objects.

- In the current version, the GBK character set supports the character €, which is represented as **0x80** in hexadecimal code. You can use the € character in the GBK library, and the GBK character set of DWS is compatible with the CP936 character set. Note that the GBK character set is approximately equal to the CP936 character set, but the GBK character set does not contain the definition of the character €.

Examples

Create database **music** using GBK (the local encoding type is also GBK).

```
CREATE DATABASE music ENCODING 'GBK' template = template0;
```

Create database **music2** and specify **jim** as its owner.

```
CREATE USER jim PASSWORD "{Password}";  
CREATE DATABASE music2 OWNER jim;
```

Create database **music3** using template **template0** and specify **jim** as its owner.

```
CREATE DATABASE music3 OWNER jim TEMPLATE template0;
```

Create a database compatible with Oracle and set the connection limit to **10**.

```
CREATE DATABASE ora_compatible_db DBCOMPATIBILITY 'ORA' CONNECTION LIMIT = 10;
```

Helpful Links

[ALTER DATABASE](#), [DROP DATABASE](#)

FAQs

For details, see "FAQs" > "Database Use" > "How Do I View and Set the Character Set Encoding Format of a DWS Database?" in *Data Warehouse Service (DWS) User Guide*.

21.13.2.2 ALTER DATABASE

Function

This command is used to modify the attributes of a database, including the database name, owner, maximum number of connections, and object isolation attribute.

Precautions

- Only the owner of a database or a system administrator has the permission to run the **ALTER DATABASE** statement. Users other than system administrators may have the following permission constraints depending on the attributes to be modified:
 - To modify the database name, you must have the CREATEDB permission.
 - To modify a database owner, you must be a database owner and a member of the new owner, and have the CREATEDB permission.

- To change the default tablespace, you must be a database owner or a system administrator, and must have the CREATE permission on the new tablespace. This statement physically migrates tables and indexes in a default tablespace to a new tablespace. Note that tables and indexes outside the default tablespace are not affected.
- Only a database owner or a system administrator can modify GUC parameters for the database.
- Only database owners and system administrators can modify the object isolation attribute of a database.
- You are not allowed to rename a database in use. To rename it, connect to another database.
- The compatibility mode of an existing database cannot be changed. You can only specify the compatibility mode when creating a database. For details, see [CREATE DATABASE](#).

Syntax

- Modify the maximum number of connections of the database.

```
ALTER DATABASE database_name  
[ [ WITH ] CONNECTION LIMIT connlimit ];
```

- Rename the database.

```
ALTER DATABASE database_name  
RENAME TO new_name;
```

NOTE

If the database contains OBS multi-temperature tables, the database name cannot be changed.

- Change the database owner.

```
ALTER DATABASE database_name  
OWNER TO new_owner;
```

- Modify the session parameter value of the database.

```
ALTER DATABASE database_name  
SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
```

- Reset the database configuration parameter.

```
ALTER DATABASE database_name RESET  
{ configuration_parameter | ALL };
```

- Modify the object isolation attribute of a database.

```
ALTER DATABASE database_name [ WITH ] { ENABLE | DISABLE } PRIVATE OBJECT;
```

NOTE

- To modify the object isolation attribute of a database, the database must be connected. Otherwise, the modification will fail.
- For a new database, the object isolation attribute is disabled by default. After this attribute is enabled, common users can view only the objects (such as tables, functions, views, and columns) that they have the permission to access. This attribute does not take effect for administrators. After this attribute is enabled, administrators can still view all database objects.

Parameter Description

Table 21-120 ALTER DATABASE parameters

| Parameter | Description | Value Range |
|-------------------------------|---|--|
| database_name | Specifies the name of the database whose attributes are to be modified. | A string compliant with the identifier naming rules . |
| connlimit | Specifies the maximum number of concurrent connections that can be made to this database (excluding administrators' connections). | An integer ranging from 1 to 50 is recommended.
The default value -1 indicates no restrictions. |
| new_name | Specifies the new name of a database. | A string compliant with the identifier naming rules . |
| new_owner | Specifies the new owner of a database. | A string that indicates a valid username. |
| configuration_parameter value | Sets a specified database session parameter to a specified value. | The value should be a string. <ul style="list-style-type: none">• DEFAULT indicates that the default settings of the system are used in a new session.• RESET indicates that the default settings of the system are used in a new session.• OFF indicates that the setting is disabled. |
| FROM CURRENT | Sets the value based on the database connected to the current session. | - |
| RESET configuration_parameter | Resets the specified database session parameter. | - |
| RESET ALL | Resets all database session parameters.
The modified database session parameter values will take effect in the next session. | - |

Examples

Create the **testdb** database and the **user1** user.

```
CREATE DATABASE testdb ENCODING 'UTF8' template = template0;  
DROP USER IF EXISTS user1;  
CREATE USER user1 PASSWORD '{Password}';
```

Set the number of connections of the **testdb** database to **10**.

```
ALTER DATABASE testdb CONNECTION LIMIT= 10;
```

Change the database name from **testdb** to **testdb_1**.

```
ALTER DATABASE testdb RENAME TO testdb_1;
```

Change the owner of the **testdb_1** database to **user1**.

```
ALTER DATABASE testdb_1 OWNER TO user1;
```

Disable the default index scan on the **testdb_1** database.

```
ALTER DATABASE testdb_1 SET enable_indexscan TO off;
```

Reset parameter **enable_indexscan**.

```
ALTER DATABASE testdb_1 RESET enable_indexscan;
```

Links

[CREATE DATABASE](#) and [DROP DATABASE](#)

21.13.2.3 DROP DATABASE

Function

Delete the database.

Precautions

- Only the owner of a database or a system administrator has the permission to run the **DROP DATABASE** command.
- **DROP DATABASE** does not take effect for the three preinstalled system databases (**gaussdb**, **TEMPLATE0**, and **TEMPLATE1**) because they are protected. To check databases in the current service, run the **\l** command of **gsql**.
- This command cannot be run while the database to be deleted is associated with a user. You can check the current database connections in the **v\$session** view.
- **DROP DATABASE** cannot be run inside a transaction block.
- If **DROP DATABASE** fails to be run and is rolled back, run **DROP DATABASE IF EXISTS**.
- If a "database is being accessed by other users" error is displayed when you run **DROP DATABASE**, it might be that threads cannot respond to signals in a timely manner during the **CLEAN CONNECTION** process. As a result, connections are not completely cleared. In this case, you need to run **CLEAN CONNECTION** again.

⚠ DANGER

- Be cautious when using **DROP OBJECT** (e.g., **DATABASE, USER/ROLE, SCHEMA, TABLE, VIEW**) as it may cause data loss, especially with **CASCADE** deletions. Always back up data before proceeding.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

Syntax

```
DROP DATABASE [ IF EXISTS ] database_name ;
```

Parameter Description

Table 21-121 Parameter Description

| Parameter | Description | Value Range |
|---------------|---|--------------------------------------|
| IF EXISTS | Sends a message instead of an error if the specified database does not exist. | - |
| database_name | Specifies the name of the database to be deleted. | Specifies an existing database name. |

Examples

Delete the database named **music**.

```
DROP DATABASE music;
```

Links

[CREATE DATABASE, ALTER DATABASE](#)

FAQs

If you see this error while running the **DROP DATABASE** command, another user is connected to the database.

database "mysql_compatible_db" is being accessed by other users Detail: There are 2 other sessions using the database.

Disconnect all users from the database.

```
CLEAN CONNECTION TO ALL FORCE FOR DATABASE mysql_compatible_db;
```

Retry the **DROP DATABASE** command—it should work. If it does not, perform **CLEAN CONNECTION** and attempt again.

```
DROP DATABASE mysql_compatible_db;
```

21.13.2.4 CREATE SCHEMA

Function

CREATE SCHEMA creates a schema.

Named objects are accessed either by "qualifying" their names with the schema name as a prefix, or by setting a search path that includes the desired schema(s). When creating named objects, you can also use the schema name as a prefix.

Optionally, **CREATE SCHEMA** can include sub-commands to create objects within the new schema. The sub-commands are treated essentially the same as separate commands issued after creating the schema. If the **AUTHORIZATION** clause is used, all the created objects are owned by this user.

Precautions

- As long as the current user has the CREATE permission, the user can create a schema.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not the system administrator.

Syntax

- Create a schema based on a specified name:

```
CREATE SCHEMA schema_name  
  [ AUTHORIZATION user_name ] [ WITH PERM SPACE 'space_limit' ] [ schema_element [ ... ] ];  
CREATE SCHEMA IF NOT EXISTS schema_name  
  [ AUTHORIZATION user_name ] [ WITH PERM SPACE 'space_limit' ] ;
```
- Create a schema based on a user name:

```
CREATE SCHEMA AUTHORIZATION user_name [ WITH PERM SPACE 'space_limit' ] [ schema_element  
  [ ... ] ];  
CREATE SCHEMA IF NOT EXISTS AUTHORIZATION user_name [ WITH PERM SPACE 'space_limit' ] ;
```

Parameter Description

Table 21-122 CREATE SCHEMA parameters

| Parameter | Description | Value Range |
|-------------|---|---|
| schema_name | Indicates the name of the schema to be created.
NOTICE <ul style="list-style-type: none">• The name must be unique,• and cannot start with pg_. | A string, which must comply with the identifier naming convention. For details, see Identifier Naming Conventions . |

| Parameter | Description | Value Range |
|-------------------------------------|--|--|
| AUTHORIZATION
user_name | Indicates the name of the user who will own this schema. If schema_name is not specified, user_name will be used as the schema name. In this case, user_name can only be a role name. | Name of an existing user or role. |
| WITH PERM
SPACE
'space_limit' | Indicates the storage upper limit of the permanent table in the specified schema. If space_limit is not specified, the space is not limited. | A string consists of an integer and unit. The unit can be K/M/G/T/P currently. The unit of parsed value is K and cannot exceed the range that can be expressed in 64 bits, which is 1 KB to 9007199254740991 KB. |
| schema_element | Indicates an SQL statement defining an object to be created within the schema. Currently, only CREATE TABLE , CREATE VIEW , CREATE INDEX , CREATE PARTITION , and GRANT are accepted as clauses within CREATE SCHEMA . Objects created by sub-commands are owned by the user specified by AUTHORIZATION . | - |

| Parameter | Description | Value Range |
|---------------|--|---|
| IF NOT EXISTS | <p>If IF NOT EXISTS is specified and no schema with the same name exists, a schema can be created. If a schema with the same name already exists during schema creation, the system will not report an error but will display a message indicating that the schema already exists and no further operations will be performed. When this parameter is used, the Schema_element subcommand cannot be included.</p> <p>The IF NOT EXISTS parameter is supported only in 9.1.0 and later versions.</p> | <p>If objects in the schema on the current search path are with the same name, specify the schemas different objects are in. You can run the SHOW SEARCH_PATH command to check the schemas on the current search path.</p> |

Examples

Create the role **role1**.

```
CREATE ROLE role1 IDENTIFIED BY '{password}';
```

-- Create the schema **role1** for the role **role1**. The owner of the **films** and **winners** tables created by the clause is **role1**.

```
CREATE SCHEMA AUTHORIZATION role1
  CREATE TABLE films (title text, release date, awards text[])
  CREATE VIEW winners AS
  SELECT title, release FROM films WHERE awards IS NOT NULL;
```

Helpful Links

[ALTER SCHEMA](#), [DROP SCHEMA](#)

21.13.2.5 ALTER SCHEMA

Function

ALTER SCHEMA changes the attributes of a schema.

Precautions

- Only the owner of a schema or users with the **ALTER** permission for the schema can run the **ALTER SCHEMA** statement. System administrators have this permission by default.
- To change the owner of a schema or its storage limit, a non-admin user must be directly or indirectly part of the new role and have **CREATE** permission on the database.

Syntax

- Rename a schema.

```
ALTER SCHEMA schema_name  
  RENAME TO new_name;
```
- Changes the owner of a schema.

```
ALTER SCHEMA schema_name  
  OWNER TO new_owner;
```
- Changes the storage space limit of the permanent table in the schema.

```
ALTER SCHEMA schema_name  
  WITH PERM SPACE 'space_limit';
```

Parameter Description

Table 21-123 ALTER SCHEMA parameters

| Parameter | Description | Value Range |
|-------------------------------|---|--|
| schema_name | Specifies the name of the schema to be modified. | Name of an existing schema. |
| new_name | Specifies the new schema name. | A string compliant with the identifier naming rules . |
| new_owner | Specifies the new schema owner. | Name of an existing user or role. |
| WITH PERM SPACE 'space_limit' | The upper limit of the permanent table storage space of the schema. | A string consists of an integer and unit. The unit can be K/M/G/T/P. The parsed value is in kilobytes (K) and must stay within the 1 KB to 9,007,199,254,740,991 KB range. |

Examples

Create an example schema **schema_test** and a user **user_a**.

```
CREATE SCHEMA schema_test;  
CREATE USER user_a PASSWORD '{Password}';
```

Rename **schema_test** to **schema_test1**.

```
ALTER SCHEMA schema_test RENAME TO schema_test1;
```

Change the owner of **schema_test1** to **user_a**.

```
ALTER SCHEMA schema_test1 OWNER TO user_a;
```

Helpful Links

[CREATE SCHEMA](#) and [DROP SCHEMA](#)

21.13.2.6 DROP SCHEMA

Function

DROP SCHEMA deletes a schema in a database.

Precautions

- Only the owner of a schema or a user granted with the DROP permission for the schema or a system administrator has the permission to execute the **Drop SCHEMA** statement.



DANGER

- Be cautious when using **DROP OBJECT** (e.g., **DATABASE, USER/ROLE, SCHEMA, TABLE, VIEW**) as it may cause data loss, especially with **CASCADE** deletions. Always back up data before proceeding.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

Syntax

```
DROP SCHEMA [ IF EXISTS ] schema_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

Table 21-124 DROP SCHEMA parameters

| Parameter | Description | Value Range |
|-------------|--|--------------------------|
| IF EXISTS | Sends a notice instead of an error if the specified schema does not exist. | - |
| schema_name | Specifies the name of the schema to be deleted. | An existing schema name. |

| Parameter | Description | Value Range |
|--------------------|---|---|
| CASCADE RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none">• CASCADE: automatically deletes all objects that are contained in the schema to be deleted.• RESTRICT: refuses to delete the schema that contains any objects. This is the default. |

NOTICE

- Do not delete the schemas with the beginning of **pg_temp** or **pg_toast_temp**. They are internal system schemas, and deleting them may cause unexpected errors.
- A user cannot delete the schema in use. To delete the schema in use, switch to another schema.

Example

Delete **ds_new** schema.

```
DROP SCHEMA ds_new;
```

Links

[ALTER SCHEMA](#), [CREATE SCHEMA](#)

21.13.2.7 CREATE EXTERNAL SCHEMA

Function

Creates an EXTERNAL schema. This syntax is supported only in 8.3.0 and later versions.

You can create an external schema to access the LakeFormation, DLI, and Hive MetaStore tables. You can use an external schema name as the prefix for access. If there is no schema name prefix, you can access the named objects in the current schema.

Precautions

- A user who has the CREATE permission on the current database can create a foreign schema.
- When creating a named object, do not use EXTERNAL SCHEMA as the prefix. Objects cannot be created in EXTERNAL SCHEMA.
- **CREATE EXTERNAL SCHEMA** does not support subcommands for creating objects in the new schema.

Syntax

- Create an external schema with a specified name.

```
CREATE EXTERNAL SCHEMA schema_name
WITH SOURCE source_name
    DATABASE 'database_name'
    SERVER server_name
    [ CATALOG 'catalog_name' ]
    [ OPTIONS ( { option_name 'value ' } [, ...] ) ]
    [METAADDRESS 'address']
    [CONFIGURATION 'confpath'];
```

Parameter Description

Table 21-125 CREATE EXTERNAL SCHEMA parameters

| Parameter | Description | Value Range |
|-------------|---|--|
| schema_name | Name of an external schema. <ul style="list-style-type: none">The name must be unique.The name cannot start with pg_. | A string, which must comply with Identifier Naming Conventions . |
| SOURCE | Type of the external metadata storage engine. | Currently, only dli , lakeformation , and hive are supported. |
| DATABASE | Database corresponding to the external schema. | - |
| SERVER | Name of the external server associated with the external schema. External data can be accessed after the association. | Servers whose type is lf , obs , dli , or hdfs can be associated. For details, see CREATE SERVER . |
| CATALOG | Catalog to be accessed in LakeFormation. This parameter is mandatory only when server type is set to lf . | - |

| Parameter | Description | Value Range |
|-------------------|---|--|
| OPTIONS | Specifies the parameters for a foreign table are listed in the right column. This function is supported only by 8.3.0 and later versions. | dli_project_id: Specifies the project ID corresponding to DLI. You can obtain the project ID from the management console. This parameter is available only when the server type is DLI. |
| METASTORE ADDRESS | Hive Metastore communication interface. This parameter is supported only by 9.1.0 and later versions. | - |
| CONFIGURATION | Path for storing Hive Metastore configuration files. This parameter is supported only by 9.1.0 and later versions. | If objects in the schema on the current search path are with the same name, specify the schemas different objects are in. You can run the SHOW SEARCH_PATH command to check the schemas on the current search path. |

Example: Read the LakeFormation Table Using the External Schema

Read the LakeFormation table using the external schema.

1. Create **lf_server**. The corresponding foreign data wrapper is **DFS_FDW**.

NOTE

For details about how to create **lf_server**, see section "Managing LakeFormation Data Sources" in *Data Warehouse Service (DWS) User Guide*.

2. Create an external schema.

Set **SOURCE** to **lakeformation**, the DLI server associated with the table to **lf_server**, and **DATABASE** to the corresponding LakeFormation database.

Set **CATALOG** to the catalog of LakeFormation to be accessed. Replace it as needed.

```
CREATE EXTERNAL SCHEMA ex_lf
WITH SOURCE lakeformation
DATABASE 'demo'
SERVER lf_server
CATALOG 'hive';
```

3. Grant permissions to a role.

- a. Query the current user.

```
SELECT current_user;
```

- b. Create a role that matches the current role on the LakeFormation management plane and assign it the table access permissions.
4. Query data in the LakeFormation table using the external schema. **test_lf** indicates the LakeFormation table to be accessed.

```
SELECT COUNT(*) FROM ex_dli.test_lf;
count
-----
      20
(1 row)
```

Reading the Hive Metastore Table Using the External Schema

Read the Hive Metastore table using the external schema. Only 9.1.0 and later versions support this function.

1. Create **obs/hdfs server**, with **DFS_FDW** as the foreign data wrapper.

```
CREATE SERVER hdfs_server
FOREIGN DATA WRAPPER HDFS_FDW
OPTIONS (
  address '***.***.***.***.9000',
  type'HDFS');

CREATE SERVER obs_server
FOREIGN DATA WRAPPER dfs_fdw
OPTIONS (
  address 'obs.example.com',
  ACCESS_KEY 'access_key_value_to_be_replaced',
  SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
  encrypt 'on',
  type 'obs');
```

2. Create an external schema. Set **SOURCE** to **hive** and the server associated with the table to **obs/hdfs server**. **DATABASE** indicates the HMS database, **METAADDRESS** indicates the Hive Metastore communication interface, and **CONFIGURATION** is an absolute path for storing Hive Metastore configuration files. Adjust these parameters as needed.

Before manually creating an external schema to interconnect with Hive Metastore, you need to:

- a. Obtain the **hive-site.xml** and **hivemetastore-site.xml** configuration files from the cluster management page or the Hive installation directory, and upload them to the **CONFIGURATION** directory.
- b. Add the **elk-site.xml** configuration file and configuration items (or add configuration items to the original **elk-site.xml** configuration file). Replace the username, domain name, and keytab file name as required.

```
<property>
  <name>hive.kerberos.principal</name>
  <value>user/hadoop.hadoop.com@HADOOP.COM</value> //User for connecting to
Hadoop@Domain name
</property>
<property>
  <name>hive.kerberos.keytab</name>
  <value>{Configuration directory}/user.keytab</value> // Absolute path of the keytab
file of the user for connecting to Hadoop
</property>
```

```
CREATE EXTERNAL SCHEMA ex_hms
WITH SOURCE source_type
DATABASE 'db_name'
SERVER srv_name
METAADDRESS 'address'
CONFIGURATION 'confpath';
```

- Query data in the HMS table using the external schema. **test_hms** indicates the HMS table to be accessed. Replace it as needed.

```
SELECT COUNT(*) FROM ex_hms.test_hms;
count
-----
      20
(1 row)
```

Helpful Links

[ALTER EXTERNAL SCHEMA](#)

21.13.2.8 ALTER EXTERNAL SCHEMA

Function

Modifies EXTERNAL SCHEMA. This syntax is supported only in 8.3.0 and later versions.

Syntax

Modify an external schema based on the specified name.

```
ALTER EXTERNAL SCHEMA schema_name
WITH [ SOURCE source_name ]
    [ DATABASE 'database_name' ]
    [ SERVER server_name ]
    [ CATALOG 'catalog_name' ]
    [ OPTIONS ( { option_name 'value ' } [, ...] ) ]
    [ METAADDRESS 'address']
    [ CONFIGURATION 'confpath'];
```

Parameter Description

Table 21-126 ALTER EXTERNAL SCHEMA parameters

| Parameter | Description | Value Range |
|-------------|---|---|
| schema_name | Specifies the name of the external schema to be modified. | Name of an existing schema. |
| SOURCE | Specifies the type of the external metadata storage engine. | Currently, source_type can only be set to dli , lakeformation , or hive . |
| DATABASE | Specifies the database to be accessed by the external schema. | - |
| SERVER | You can associate an external schema with a foreign server to access external data. | An existing foreign server whose type is lf , obs , dli , or hdfs . |
| CATALOG | Catalog to be accessed in LakeFormation. | - |

| Parameter | Description | Value Range |
|---------------|--|---|
| OPTIONS | Specifies parameters of foreign table data. This is supported only by 8.3.0 and later versions. | dli_project_id : project ID of DLI. You can obtain the project ID from the management console. This parameter can be set only when the server type is DLI. |
| METAADDRESS | Specifies the Hive Metastore communication port. This is supported only by 9.1.0 and later versions. | - |
| CONFIGURATION | Specifies the path for storing Hive Metastore configuration files. This parameter is supported only by clusters of version 9.1.0 or later.

If objects in the schema on the current search path are with the same name, specify the schemas different objects are in. You can run the SHOW SEARCH_PATH command to check the schemas on the current search path. | - |

Examples

Modify the database and **FOREIGN SERVER** corresponding to the **EXTERNAL SCHEMA** whose name is **ex1**.

```
ALTER EXTERNAL SCHEMA ex1
  WITH DATABASE 'demo'
  SERVER my_server;
```

21.13.2.9 CREATE TYPE

Function

CREATE TYPE defines new data types in the current database. The user who defines a new data type becomes its owner. Types are designed only for row-store tables.

Four types of data can be created by using **CREATE TYPE**: composite data, base data, a shell data, and enumerated data.

- Composite types

A composite type is specified by a list of attribute names and data types. If the data type of an attribute is collatable, the attribute's collation rule can also be specified. A composite type is essentially the same as the row type of a table. However, using **CREATE TYPE** avoids the need to create an actual

table when only a type needs to be defined. In addition, a standalone composite type is useful, for example, as the parameter or return type of a function.

To create a composite type, you must have the **USAGE** permission for all its attribute types.

- Base types

You can customize a new base type (scalar type). Generally, functions required for base types must be coded in C or another low-level language.

- Shell types

With this parameter, you can create a shell type, which is a type name without a definition. To create a shell type, use **CREATE TYPE** with only the type name. Shell types are needed as forward references when base types are created.

- Enumerated types

An enumerated type is a list of enumerated values. Each value is a non-empty string with the maximum length of 64 bytes.

Precautions

- If a schema name is given, the type will be created in the specified schema. Otherwise, it will be created in the current schema. The type name must be distinct from the name of any existing type or domain in the same schema. (Because tables have associated data types, the type name must also be distinct from the name of any existing table in the same schema.)
- When creating a base type, you can place parameters in any order. The **input_function** and **output_function** parameters are mandatory, and other parameters are optional.
- The **category** and **preferred** parameters can be used to help determine which implicit cast excels in ambiguous situations. Each data type belongs to a category named by a single ASCII character, and each type is either preferred or not within its category. If this rule is helpful in resolving overloaded functions or operators, the parser will prefer casting to preferred types (but only from other types within the same category). For types that have no implicit casts to or from any other types, it is sufficient to leave these parameters at their default values. However, for a group of types that have implicit casts, mark them all as belonging to a category and select one or two of the most general types as being preferred within the category. The **category** parameter is helpful in adding a user-defined type to an existing built-in category, such as the numeric or string type. However, you can also create new entirely-user-defined type categories. Select any ASCII character other than an uppercase letter to name such a category.
- **type_modifier_input_function** and **type_modifier_output_function** are needed if a type supports modifiers, that is, optional constraints attached to a type declaration, such as `char(5)` or `numeric(30,2)`. DWS allows user-defined types to take one or more simple constants or identifiers as modifiers. However, this information must be capable of being packed into a single non-negative integer value for storage in system catalogs. Declared modifiers are passed to **type_modifier_input_function** in the cstring array format. The parameter must check values for validity, throwing an error if they are wrong. If they are correct, the parameter will return a single non-negative integer

value, which will be stored as `typmod` in a column. If the type does not have **type_modifier_input_function**, type modifiers will be rejected.

type_modifier_output_function converts the internal integer `typmod` value back to a correct format for user display. It must return a `cstring` value, which is the exact string appending to the type name. For example, a numeric function may return (30,2). If the default display format is enclosing a stored `typmod` integer value in parentheses, you can omit **type_modifier_output_function**.

- Input and output functions can be declared to have the results or parameters of a new type because they have to be created before the new type is created. The new type should first be defined as a shell type, which is a placeholder type that has no attributes except a name and an owner. This can be done by delivering the **CREATE TYPE** *name* statement, with no additional parameters. Then, the C I/O functions can be defined as referencing the shell type. Finally, **CREATE TYPE** with a full definition replaces the shell type with a complete, valid type definition. After that, the new type can be used normally.

Syntax

```
CREATE TYPE name AS
    ( [ attribute_name data_type [ COLLATE collation ] [, ... ] ] )

CREATE TYPE name (
    INPUT = input_function,
    OUTPUT = output_function
    [ , RECEIVE = receive_function ]
    [ , SEND = send_function ]
    [ , TYPMOD_IN = type_modifier_input_function ]
    [ , TYPMOD_OUT = type_modifier_output_function ]
    [ , ANALYZE = analyze_function ]
    [ , INTERNALLENGTH = { internallength | VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = alignment ]
    [ , STORAGE = storage ]
    [ , LIKE = like_type ]
    [ , CATEGORY = category ]
    [ , PREFERRED = preferred ]
    [ , DEFAULT = default ]
    [ , ELEMENT = element ]
    [ , DELIMITER = delimiter ]
    [ , COLLATABLE = collatable ]
)

CREATE TYPE name

CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )
```

Parameter Description

Table 21-127 CREATE TYPE parameters

| Parameter | Description | Value Range |
|----------------|---|---|
| name | Specifies the name (optionally schema-qualified) of the type to be created. | It must comply with the naming convention . |
| attribute_name | Specifies the name of an attribute (column) for the composite type. | - |
| data_type | Specifies the name of an existing data type to become a column of the composite type. | - |
| collation | Specifies the name of an existing collation associated with a column of the composite type. | - |

| Parameter | Description | Value Range |
|-----------------|--|-------------|
| input_function | <p>Specifies the name of a function that converts data from the external text format of a type to its internal format.</p> <p>An input function can be declared as taking one parameter of the cstring type or taking three parameters of the cstring, oid, and integer types.</p> <ul style="list-style-type: none">• The cstring-type parameter is the input text as a C string.• The oid-type parameter is the OID of the type (except for array types, where the parameter is the element type OID of an array type).• The integer-type parameter is typmod of the destination column, if known (-1 will be passed if not known). <p>An input function must return a value of the data type itself. Generally, an input function must be declared as STRICT. If it is not, it will be called with a NULL parameter coming first when the system reads a NULL input value. In this case, the function must still return NULL unless an error raises. (This mechanism is designed for supporting domain input functions, which may need to reject NULL input values.)</p> | - |
| output_function | <p>Specifies the name of a function that converts data from the internal format of a type to its external text format.</p> <p>An output function must be declared as taking one parameter of a new data type. It must return data of the cstring type. Output functions are not invoked for NULL values.</p> | - |

| Parameter | Description | Value Range |
|---|---|-------------|
| receive_function (optional) | <p>Specifies the name of a function that converts data from the external binary format of a type to its internal format.</p> <p>If this function is not used, the type cannot participate in binary input. It costs lower to convert the binary format to the internal format, more portable. (For example, the standard integer data types use the network byte order as an external binary representation, whereas the internal representation is in the machine's native byte order.) This function should perform adequate checks to ensure a valid value.</p> <p>Also, this function can be declared as taking one parameter of the internal type or taking three parameters of the internal, oid, and integer types.</p> <ul style="list-style-type: none">• The internal-type parameter is a pointer to a StringInfo buffer holding received byte strings.• The oid- and integer-type parameters are the same as those of the text input function. <p>A receive function must return a value of the data type itself. Generally, a receive function must be declared as STRICT. If it is not, it will be called with a NULL parameter coming first when the system reads a NULL input value. In this case, the function must still return NULL unless an error raises. (This mechanism is designed for supporting domain receive functions, which may need to reject NULL input values.)</p> | - |
| send_function (optional) | <p>Specifies the name of a function that converts data from the internal format of a type to its external binary format.</p> <p>If this function is not used, the type cannot participate in binary output. A send function must be declared as taking one parameter of a new data type. It must return data of the bytea type. Send functions are not invoked for NULL values.</p> | - |
| type_modifier_input_function (optional) | <p>Specifies the name of a function that converts an array of modifiers for a type to its internal format.</p> | - |

| Parameter | Description | Value Range |
|--|--|-------------|
| type_modifier_output_function (optional) | Specifies the name of a function that converts the internal format of modifiers for a type to its external text format. | - |
| analyze_function (optional) | Specifies the name of a function that performs statistical analysis for a data type.

By default, if there is a default B-tree operator class for a type, ANALYZE will attempt to gather statistics by using the "equals" and "less-than" operators of the type. This behavior is inappropriate for non-scalar types, and can be overridden by specifying a custom analysis function. The analysis function must be declared to take one parameter of the internal type and return a boolean result. | - |
| internallength (optional) | Specifies a numeric constant for specifying the length in bytes of the internal representation of a new type. By default, it is variable-length.

Although the details of the new type's internal representation are only known to I/O functions and other functions that you create to work with the type, there are still some attributes of the internal representation that must be declared to DWS. The most important one is internallength . Base data types can be fixed-length (when internallength is a positive integer) or variable-length (when internallength is set to VARIABLE ; internally, this is represented by setting typlen to -1). The internal representation of all variable-length types must start with a 4-byte integer. internallength defines the total length. | - |
| PASSEDBYVALUE (optional) | Specifies that values of a data type are passed by value, rather than by reference. Types passed by value must be fixed-length, and their internal representation cannot be larger than the size of the Datum type (4 bytes on some machines, and 8 bytes on others). | - |
| alignment (optional) | Specifies the storage alignment required for a data type. It supports values char , int2 , int4 , and double . The default value is int4 .

The allowed values equate to alignment on 1-, 2-, 4-, or 8-byte boundaries. Note that variable-length types must have an alignment of at least 4 since they must contain an int4 value as their first component. | - |

| Parameter | Description | Value Range |
|----------------------|---|-------------|
| storage (optional) | <p>Specifies the storage strategy for a data type. It supports values plain, external, extended, and main. The default value is plain.</p> <ul style="list-style-type: none">• plain specifies that data of a type will always be stored in-line and not compressed. (Only plain is allowed for fixed-length types.)• extended specifies that the system will first try to compress a long data value and will then move the value out of the main table row if it is still too long.• external allows a value to be moved out of the main table, but the system will not try to compress it.• main allows for compression, but discourages moving a value out of the main table. (Data items with this storage strategy might still be moved out of the main table if there is no other way to make a row fit. However, they will be kept in the main table preferentially over extended and external items.) <p>All storage values except plain imply that the functions of the data type can handle values that have been toasted. A specified value determines the default TOAST storage strategy for columns of data types that support TOAST. Users can select different strategies for individual columns using ALTER TABLE SET STORAGE.</p> | - |
| like_type (optional) | <p>Specifies the name of an existing data type that has the same representation as a new type. The values of internallength, passedbyvalue, alignment, and storage are copied from this type, unless they are overridden by explicit specifications elsewhere in the CREATE TYPE command.</p> <p>Specifying representation in this way is especially useful when the low-level implementation of a new type references an existing type.</p> | - |

| Parameter | Description | Value Range |
|-----------------------|--|-------------|
| category (optional) | Specifies the category code (a single ASCII character) for a type. The default value is U, indicating the user-defined type. You can also choose other ASCII characters to create custom categories. | - |
| preferred (optional) | Specifies whether a type is preferred within its type category. If it is, the value will be TRUE , else FALSE . The default value is FALSE . Be cautious when creating a new preferred type within an existing type category because this could cause great changes in behavior. | - |
| default (optional) | <p>Specifies the default value for a data type. If this parameter is omitted, the default value will be NULL.</p> <p>A default value can be specified if you expect the columns of a data type to default to something other than the NULL value. You can also specify a default value using the DEFAULT keyword. (Such a default value can be overridden by an explicit DEFAULT clause attached to a particular column.)</p> | - |
| element (optional) | Specifies the type of an array element when an array type is created. For example, to define an array of 4-byte integers (int4), set ELEMENT to int4 . | - |
| delimiter (optional) | <p>Specifies the delimiter character to be used between values in arrays made of a type.</p> <p>delimiter can be set to a specific character. The default delimiter is a comma (,). Note that a delimiter is associated with the array element type, instead of the array type itself.</p> | - |
| collatable (optional) | <p>Specifies whether a type's operations can use collation information. If they can, the value will be TRUE, else FALSE (default).</p> <p>If collatable is TRUE, column definitions and expressions of a type may carry collation information by using the COLLATE clause. It is the implementations of functions operating on the type that actually use the collation information. This use cannot be achieved merely by marking the type collatable.</p> | - |

| Parameter | Description | Value Range |
|------------------|--|-------------|
| label (optional) | Specifies a text label associated with an enumerated value. It is a non-empty string of up to 64 characters. | - |

Examples

Example 1: Create a composite type, create a table, insert data, and make a query.

```
CREATE TYPE compfoo AS (f1 int, f2 text);
CREATE TABLE t1_compfoo(a int, b compfoo);
CREATE TABLE t2_compfoo(a int, b compfoo);
INSERT INTO t1_compfoo values(1,(1,'demo'));
INSERT INTO t2_compfoo select * from t1_compfoo;
SELECT (b).f1 FROM t1_compfoo;
SELECT * FROM t1_compfoo t1 join t2_compfoo t2 on (t1.b).f1=(t1.b).f1;
```

Example 2: Create an enumeration type and use it in the table definition.

```
CREATE TYPE bugstatus AS ENUM ('create', 'modify', 'closed');
CREATE TABLE customer (name text,current_bugstatus bugstatus);
INSERT INTO customer VALUES ('type','create');
SELECT * FROM customer WHERE current_bugstatus = 'create';
```

Example 3: Compile a .so file and create the shell type.

```
CREATE TYPE complex;
```

This statement creates a placeholder for the type to be created, which can then be referenced when defining its I/O functions. Now you can define an I/O function. Note that the function must be declared in NOT FENCED mode when it is created.

```
CREATE FUNCTION
complex_in(cstring)
  RETURNS complex
  AS 'filename'
  LANGUAGE C IMMUTABLE STRICT not fenced;

CREATE FUNCTION
complex_out(complex)
  RETURNS cstring
  AS 'filename'
  LANGUAGE C IMMUTABLE STRICT not fenced;

CREATE FUNCTION
complex_rcv(internal)
  RETURNS complex
  AS 'filename'
  LANGUAGE C IMMUTABLE STRICT not fenced;

CREATE FUNCTION
complex_send(complex)
  RETURNS bytea
  AS 'filename'
  LANGUAGE C IMMUTABLE STRICT not fenced;
```

Provide a complete definition of the data type.

```
CREATE TYPE complex (
internallength = 16,
```

```
input = complex_in,  
output = complex_out,  
receive = complex_rcv,  
send = complex_send,  
alignment = double  
);
```

The C functions corresponding to the input, output, receive, and send functions are defined as follows:

```
-- Define a structure body Complex:  
typedef struct Complex {  
    double    x;  
    double    y;  
} Complex;  
  
-- Define an input function:  
PG_FUNCTION_INFO_V1(complex_in);  
  
Datum  
complex_in(PG_FUNCTION_ARGS)  
{  
    char      *str = PG_GETARG_CSTRING(0);  
    double    x,  
             y;  
    Complex   *result;  
  
    if (sscanf(str, " ( %lf , %lf )", &x, &y) != 2)  
        ereport(ERROR,  
                (errmsg(ERRCODE_INVALID_TEXT_REPRESENTATION),  
                 errmsg("invalid input syntax for complex: \"%s\"",  
                        str)));  
  
    result = (Complex *) palloc(sizeof(Complex));  
    result->x = x;  
    result->y = y;  
    PG_RETURN_POINTER(result);  
}  
  
-- Define an output function:  
PG_FUNCTION_INFO_V1(complex_out);  
  
Datum  
complex_out(PG_FUNCTION_ARGS)  
{  
    Complex   *complex = (Complex *) PG_GETARG_POINTER(0);  
    char      *result;  
  
    result = (char *) palloc(100);  
    snprintf(result, 100, "(%g,%g)", complex->x, complex->y);  
    PG_RETURN_CSTRING(result);  
}  
  
-- Define a receive function:  
PG_FUNCTION_INFO_V1(complex_rcv);  
  
Datum  
complex_rcv(PG_FUNCTION_ARGS)  
{  
    StringInfo buf = (StringInfo) PG_GETARG_POINTER(0);  
    Complex   *result;  
  
    result = (Complex *) palloc(sizeof(Complex));  
    result->x = pq_getmsgfloat8(buf);  
    result->y = pq_getmsgfloat8(buf);  
    PG_RETURN_POINTER(result);  
}  
  
-- Define a send function:  
PG_FUNCTION_INFO_V1(complex_send);
```

```
Datum
complex_send(PG_FUNCTION_ARGS)
{
    Complex    *complex = (Complex *) PG_GETARG_POINTER(0);
    StringInfoData buf;

    pq_begintypsend(&buf);
    pq_sendfloat8(&buf, complex->x);
    pq_sendfloat8(&buf, complex->y);
    PG_RETURN_BYTEA_P(pq_endtypsend(&buf));
}
```

Helpful Links

[ALTER TYPE, DROP TYPE](#)

21.13.2.10 ALTER TYPE

Function

ALTER TYPE modifies the definition of a type.

Precautions

- Only type owners can run **ALTER TYPE**.
- To modify the schema of a type, you must also have the **CREATE** permission for the new schema.
- To change the owner, you must belong directly or indirectly to the new owner's role. You also need the **CREATE** permission on the schema of the type.
- System administrators can change the ownership of any type as needed.
- To add an attribute or modify the type of an attribute, you must have the **USAGE** permission on the attribute type.

Syntax

- **Modify a type.**

```
ALTER TYPE name action [ ... ];
ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER };
ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE | RESTRICT ];
ALTER TYPE name RENAME TO new_name;
ALTER TYPE name SET SCHEMA new_schema;
ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER } neighbor_enum_value ];
ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value;
```

where action is one of:

```
ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ];
DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ];
ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ CASCADE | RESTRICT ];
```
- **Add a new attribute to a composite type.**

```
ALTER TYPE name ADD ATTRIBUTE attribute_name data_type [ COLLATE collation ] [ CASCADE | RESTRICT ];
```
- **Delete an attribute from a composite type.**

```
ALTER TYPE name DROP ATTRIBUTE [ IF EXISTS ] attribute_name [ CASCADE | RESTRICT ];
```

- Change the type of an attribute in a composite type.
`ALTER TYPE name ALTER ATTRIBUTE attribute_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ CASCADE | RESTRICT ];`

NOTE

ADD ATTRIBUTE, DROP ATTRIBUTE, and ALTER ATTRIBUTE can be combined for execution. For example, you can add multiple attributes to a command or change the types of multiple attributes.

- Change the owner of a type.
`ALTER TYPE name OWNER TO { new_owner | CURRENT_USER | SESSION_USER };`
- Change the name of a type or the name of an attribute in a composite type.
`ALTER TYPE name RENAME TO new_name;`
`ALTER TYPE name RENAME ATTRIBUTE attribute_name TO new_attribute_name [ CASCADE | RESTRICT ];`
- Move a type to a new schema.
`ALTER TYPE name SET SCHEMA new_schema;`
- Add a new value to an enumerated type.
`ALTER TYPE name ADD VALUE [ IF NOT EXISTS ] new_enum_value [ { BEFORE | AFTER } neighbor_enum_value ];`
- Change an enumerated value in the value list.
`ALTER TYPE name RENAME VALUE existing_enum_value TO new_enum_value;`

Parameter Description

Table 21-128 ALTER TYPE parameters

| Parameter | Description | Value Range |
|--------------------|---|--------------------|
| name | Specifies the name of an existing type to be modified, which can include a schema. | Valid type name. |
| new_name | Specifies the new name of a type. | A string. |
| new_owner | Specifies the username of the new owner of the type. | Valid username. |
| new_schema | Specifies the new schema of the type. | Valid schema name. |
| attribute_name | Specifies the name of the attribute to be added, changed, or deleted. | - |
| new_attribute_name | Specifies the new name of the property to be renamed. | - |
| data_type | Specifies the data type of the attribute to be added or the new type name of the attribute to be changed. | - |

| Parameter | Description | Value Range |
|---------------------|--|---|
| new_enum_value | Specifies the new tag value of the enumeration type. | A string. The value cannot be empty and contains a maximum of 64 bytes. |
| neighbor_enum_value | Specifies an existing enumerated value before or after which a new enumerated value will be added. | - |
| existing_enum_value | Specifies the existing enumeration value to be renamed. | A string. The value cannot be empty and contains a maximum of 64 bytes. |
| CASCADE | Determines that the type to be modified, its associated records, and subtables that inherit the type will all be updated. | - |
| RESTRICT | Refuses the update if the type of the association to be updated is an updated type. RESTRICT is the default option. | - |

Examples

Create an example composite type **test**, enumeration type **testdata**, and user **user_t**.

```
CREATE TYPE test AS (col1 int, col text);  
CREATE TYPE testdata AS ENUM ('create', 'modify', 'closed');  
CREATE USER user_t PASSWORD '{Password}';
```

Rename data type **test** to **test1**.

```
ALTER TYPE test RENAME TO test1;
```

Change the owner of the user-defined type **test1** to **user_t**.

```
ALTER TYPE test1 OWNER TO user_t;
```

Change the schema of the user-defined type **test1** to **user_t**.

```
ALTER TYPE test1 SET SCHEMA user_t;
```

Add a new attribute **col3** to the user-defined type **test1**.

```
ALTER TYPE user_t.test1 ADD ATTRIBUTE col3 int;
```

Add a tag value to the enumeration type **testdata**.

```
ALTER TYPE testdata ADD VALUE IF NOT EXISTS 'regress' BEFORE 'closed';
```

Rename a tag value of the enumeration type **testdata**.

```
ALTER TYPE testdata RENAME VALUE 'create' TO 'new';
```

Helpful Links

[CREATE TYPE](#) and [DROP TYPE](#)

21.13.2.11 DROP TYPE

Function

DROP TYPE deletes a user-defined data type. Only the type owner has permission to run this statement.

Syntax

```
DROP TYPE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Parameter Description

Table 21-129 DROP TYPE parameters

| Parameter | Description | Value Range |
|--------------------|---|--|
| IF EXISTS | Sends a notice instead of an error if the specified type does not exist. | - |
| name | Specifies the name of the type to be deleted (schema-qualified). | Specifies an existing domain type. |
| CASCADE RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none">• CASCADE: Deletes objects (such as columns, functions, and operators) that depend on the type.• RESTRICT: Refuses to delete the type if any objects depend on it. This is the default. |

Examples

Delete **compfoo** type.

```
DROP TYPE compfoo cascade;
```

Helpful Links

[ALTER TYPE](#), [CREATE TYPE](#)

21.13.3 Users and Roles

21.13.3.1 CREATE USER

Function

Create a database user.

Precautions

- A user created using the **CREATE USER** statement has the **LOGIN** permission by default.
- A schema named after the user is automatically created in the database where the statement is executed, but not in other databases. You can run the **CREATE SCHEMA** statement to create such a schema for the user in other databases.
- The owner of an object created by a system administrator in a schema with the same name as a common user is the common user, not a system administrator.
- Users who are not system administrators and are granted with all schema permissions cannot create objects in a schema named after a user, unless the users are granted with the role permissions of that schema. For details, see **After the all Permission Is Granted to the Schema of a User, the Error Message "ERROR: current user does not have privilege to role tom" Persists During Table Creation** in *Troubleshooting*.

CAUTION

- Avoid having all business operations run under a single database user. Instead, plan different database users according to the business modules.
 - For better access control of different business modules, it is better to use multiple users and permissions instead of depending on the system administrator user to run business operations.
 - For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.
-

Syntax

```
CREATE USER user_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' | DISABLE };
```

The **option** clause is used for setting information including permissions and attributes.

```
{SYSADMIN | NOSYSADMIN}  
| {AUDITADMIN | NOAUDITADMIN}  
| {CREATEDB | NOCREATEDB}  
| {USEFT | NOUSEFT}  
| {CREATEROLE | NOCREATEROLE}  
| {INHERIT | NOINHERIT}  
| {LOGIN | NOLOGIN}  
| {REPLICATION | NOREPLICATION}  
| {INDEPENDENT | NOINDEPENDENT}  
| {VCADMIN | NOVCADMIN}  
| CONNECTION LIMIT connlimit
```



```
| VALID BEGIN 'timestamp'  
| VALID UNTIL 'timestamp'  
| RESOURCE POOL 'respool'  
| USER GROUP 'groupuser'  
| PERM SPACE 'spacelimit'  
| TEMP SPACE 'tmpspacelimit'  
| SPILL SPACE 'spillspacelimit'  
| NODE GROUP logic_cluster_name  
| IN ROLE role_name [, ...]  
| IN GROUP role_name [, ...]  
| ROLE role_name [, ...]  
| ADMIN role_name [, ...]  
| USER role_name [, ...]  
| SYSID uid  
| DEFAULT TABLESPACE tablespace_name  
| PROFILE DEFAULT  
| PROFILE profile_name  
| PGUSER  
| AUTHINFO 'authinfo'  
| PASSWORD EXPIRATION period
```

Parameter Description

Table 21-130 CREATE USER parameters

| Parameter | Description | Value Range/
Default Value |
|-----------|---|---|
| user_name | Name of the user to be created. | A string. It must comply with the naming conventions and can contain a maximum of 63 characters. For details, see Identifier Naming Conventions . |
| password | Specifies the login password.
A password must: <ul style="list-style-type: none">Contain at least eight characters. This is the default length.Differ from the user name or the user name spelled backwards.Contains at least three of the following four character types: uppercase letters, lowercase letters, digits, and special characters, including: ~!@#\$\$%^&*()-_+=\[\{\}\];;<.>/?. If you use characters other than the four types, a warning is displayed, but you can still create the password. | A string |

| Parameter | Description | Value Range/
Default Value |
|------------------------------|---|---------------------------------------|
| DISABLE | By default, you can change your password unless it is disabled. Use this parameter to disable the password of a user. After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, IAM authentication, Kerberos authentication, or LDAP authentication. Only administrators can enable or disable a password. Common users cannot disable the password of an initial user. To enable a password, run ALTER USER and specify the password. | - |
| ENCRYPTED
UNENCRYPTED | Determines whether the password stored in the system will be encrypted. (If neither is specified, the password status is determined by password_encryption_type .) According to product security requirements, the password must be stored encrypted. Therefore, UNENCRYPTED is forbidden in DWS. If the password is encrypted using SHA256, it will be stored as it is, regardless of whether it is specified as ENCRYPTED or UNENCRYPTED . This is because the system cannot decrypt the specified encrypted password. This allows reloading of the encrypted password during dump/restore. | - |
| SYSADMIN
NOSYSADMIN | Determines whether a new user is a system administrator. A user with the SYSADMIN attribute has the highest permission in the system. | Default value:
NOSYSADMIN |
| AUDITADMIN
NOAUDITADMIN | Defines whether a user has the audit administrator attribute. | Default value:
NOAUDITADMIN |
| CREATEDB
NOCREATEDB | Controls whether a new user can create a database.

A new user does not have the permission to create a database by default. | Default value:
NOCREATEDB |

| Parameter | Description | Value Range/
Default Value |
|-----------------------------------|---|--|
| USEFT
NOUSEFT | Determines whether a new role can perform operations on foreign tables, such as creating, deleting, modifying, and reading/writing foreign tables.

The default value is NOUSEFT , indicating that new users cannot perform operations on foreign tables by default.

The new user does not have the permission to perform operations on foreign tables. | - |
| CREATEROLE

NOCREATEROLE | Controls whether a user can create a role or user (that is, execute CREATE ROLE and CREATE USER). A user with the CREATEROLE permission can also modify and delete other users or roles. | Value range: If not specified, NOCREATEROLE is the default. |
| INHERIT
NOINHERIT | Controls whether a user can inherit the permissions of its group. You are not advised to use them. | - |
| LOGIN
NOLOGIN | Controls whether a user can log in to the database. Only users with the LOGIN attribute can log in to the database. | Default value:
LOGIN |
| REPLICATION

NOREPLICATION | Controls whether a user is allowed to initiate streaming replication or put the system in and out of backup mode. A user with the REPLICATION attribute is only used for replication. | Default value:
NOREPLICATION |

| Parameter | Description | Value Range/
Default Value |
|-----------------------------|--|-------------------------------|
| INDEPENDENT NOINDEPENDENT | <p>Defines private and independent users. For a user with the INDEPENDENT attribute, administrators' rights to control and access this role are separated. Specific rules are as follows:</p> <ul style="list-style-type: none">Administrators have no rights to add, delete, query, modify, copy, or authorize the corresponding table objects without the authorization from the INDEPENDENT user.Without the authorization of the INDEPENDENT user, the administrator has no right to modify its inheritance relationship.The administrator does not have the permission to change the owner of the table object of an INDEPENDENT user.The administrator does not have the permission to remove the INDEPENDENT attribute of an INDEPENDENT user.The administrator does not have the permission to change the database password of an INDEPENDENT user. An INDEPENDENT must manage its own password. If the password is lost, it cannot be reset.The SYSADMIN attribute of a user cannot be changed to the INDEPENDENT attribute. | - |
| VCADMIN NOVCADMIN | <p>Defines a logical cluster administrator. A logical cluster administrator has the following more permissions than common users:</p> <ul style="list-style-type: none">Create, modify, and delete resource pools in the associated logical cluster.Grant the access permission for the associated logical cluster to other users or roles, or reclaim the access permission from those users or roles. | - |

| Parameter | Description | Value Range/
Default Value |
|---------------------------|--|--|
| CONNECTION
LIMIT | <p>Specifies the number of concurrent connections that can be used by a user on a single CN.</p> <p>To ensure that a cluster runs properly, the minimum value of connlimit is the number of CNs in the cluster, because when a cluster runs ANALYZE on a CN, other CNs will connect to the running CN for metadata synchronization. For example, if there are three CNs in the cluster, set CONNECTION LIMIT to 3 or a larger value.</p> | <p>Integers no less than -1.</p> <p>The default value -1 means no limit.</p> |
| VALID BEGIN | Sets the timestamp when a user takes effect. If this clause is omitted, there is no restriction on when the user takes effect. | - |
| VALID UNTIL | Sets an expiration timestamp for an account. After the specified time, the account cannot be used. If this clause is omitted, there is no restriction on when the user expires. | - |
| RESOURCE
POOL | Sets the name of resource pool used by a user, and the name belongs to the system catalog: pg_resource_pool . | - |
| USER GROUP
'groupuser' | Create a sub-user. | - |
| PERM SPACE | <p>Sets the storage space of the user permanent table.</p> <p>space_limit: specifies the upper limit of the storage space of the permanent table.</p> | <p>space_limit: A string consists of an integer and unit. The unit can be K/M/G/T/P currently.</p> <p>0 indicates no limits.</p> |
| TEMP SPACE | <p>Sets the storage space of the user temporary table.</p> <p>tmpspacelimit: specifies the storage space limit of the temporary table.</p> | <p>tmpspacelimit: A string consists of an integer and unit. The unit can be K/M/G/T/P currently.</p> <p>0 indicates no limits.</p> |

| Parameter | Description | Value Range/
Default Value |
|--------------------|--|--|
| SPILL SPACE | Sets the operator disk flushing space of the user.
spillspacelimit : specifies the operator spilling space limit. | spillspacelimit : A string consists of an integer and unit. The unit can be K/M/G/T/P currently.
0 indicates no limits. |
| NODE GROUP | Specifies the name of the logical cluster associated with a user. If the name contains uppercase characters or special characters, enclose the name with double quotation marks. | - |
| IN ROLE | The new user immediately has the permissions of users listed in the IN ROLE clause. You are not advised to use them. | - |
| IN GROUP | Indicates an obsolete spelling of IN ROLE . You are not advised to execute them. | - |
| ROLE | The ROLE clause lists one or more existing users. They are automatically added as members of the new user and have all the permissions of the new user. | - |
| ADMIN | The ADMIN clause is similar to the ROLE clause. The difference is that the user after ADMIN can grant the permissions of the new user to other users. | - |
| USER | Indicates an obsolete spelling of the ROLE clause. | - |
| SYSID | The SYSID clause is ignored. | - |
| DEFAULT TABLESPACE | The DEFAULT TABLESPACE clause is ignored. | - |
| PROFILE | The PROFILE clause is ignored. | - |

| Parameter | Description | Value Range/
Default Value |
|------------------------|---|-------------------------------|
| PGUSER | <p>This attribute is used to be compatible with open-source Postgres communication. An open-source Postgres client interface (Postgres 9.2.19 is recommended) can use a database user having this attribute to connect to the database.</p> <p>This attribute only ensures compatibility with the connection process. Incompatibility caused by kernel differences between this product and Postgres cannot be solved using this attribute.</p> | - |
| AUTHINFO
'authinfo' | <p>This attribute is used to specify the user authentication type. authinfo is the description character string, which is case sensitive.</p> <p>Only the LDAP type is supported. Its description character string is ldap. LDAP authentication is an external authentication mode. Therefore, PASSWORD DISABLE must be specified.</p> <ul style="list-style-type: none">Additional information about LDAP authentication can be added to authinfo, for example, fulluser in LDAP authentication, which is equivalent to ldapprefix+username+ldapsuffix. If authinfo is ldap, the user authentication type is LDAP. In this case, the ldapprefix and ldapsuffix information is provided by the corresponding record in the pg_hba.conf file.When executing the ALTER ROLE command, users are not allowed to change the authentication type. Only LDAP users are allowed to modify LDAP attributes. | - |

| Parameter | Description | Value Range/
Default Value |
|----------------------------|--|--|
| PASSWORD EXPIRATION period | Number of days before the login password of the role expires. The user needs to change the password in time before the login password expires. If the login password expires, the user cannot log in to the system. In this case, the user needs to ask the administrator to set a new login password. | The value is an integer ranging from -1 to 999. <ul style="list-style-type: none">• The default value -1 indicates no restrictions.• 0: indicates that the login password expires immediately. |

Examples

Create user **new_u1**.

```
CREATE USER new_u1 PASSWORD '{password}';
```

The following statements are equivalent to the above.

```
CREATE USER new_u1 IDENTIFIED BY '{password}';
```

For a user having the **CREATE DATABASE** permission, specify the **CREATEDB** keyword.

```
CREATE USER new_u1 CREATEDB PASSWORD '{password}';
```

Helpful Links

[ALTER USER](#), [CREATE ROLE](#), [DROP USER](#)

21.13.3.2 ALTER USER

Function

ALTER USER modifies the attributes of a database user.

Precautions

Session parameters modified by **ALTER USER** apply to a specified user and take effect in the next session.

Syntax

- Modify user rights or other information.

```
ALTER USER user_name [ [ WITH ] option [ ... ] ];
```

The **option** clause is as follows:

```
{ CREATEDB | NOCREATEDB }  
| { CREATEROLE | NOCREATEROLE }
```



```
| { INHERIT | NOINHERIT }  
| { AUDITADMIN | NOAUDITADMIN }  
| { SYSADMIN | NOSYSADMIN }  
| { USEFT | NOUSEFT }  
| { LOGIN | NOLOGIN }  
| { REPLICATION | NOREPLICATION }  
| { INDEPENDENT | NOINDEPENDENT }  
| { VCADMIN | NOVCADMIN }  
| CONNECTION LIMIT connlimit  
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE }  
| [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] |  
DISABLE }  
| VALID BEGIN 'timestamp'  
| VALID UNTIL 'timestamp'  
| RESOURCE POOL 'respool'  
| USER GROUP 'groupuser'  
| PERM SPACE 'spacelimit'  
| TEMP SPACE 'tmpspacelimit'  
| SPILL SPACE 'spillspacelimit'  
| NODE GROUP logic_cluster_name  
| ACCOUNT { LOCK | UNLOCK }  
| PGUSER  
| AUTHINFO 'authinfo'  
| PASSWORD EXPIRATION period
```

- Change the user name.
ALTER USER user_name
RENAME TO new_name;
- Change the value of a specified parameter associated with the user.
ALTER USER user_name
SET configuration_parameter { { TO | = } { value | DEFAULT } | FROM CURRENT };
- Reset the value of a specified parameter associated with the user.
ALTER USER user_name
RESET { configuration_parameter | ALL };

Parameter Description

Table 21-131 ALTER USER parameters

| Parameter | Description | Value Range/Default Value |
|---------------------------|--|---|
| user_name | Specifies the username to be changed. | Valid username. |
| new_name | Specifies the new name of a user. | A string compliant with the identifier naming rules . |
| CREATEDB NOCREATEDB | Controls whether a new user can create a database.

A new user does not have the permission to create a database by default. | Default value:
NOCREATEDB |
| CREATEROLE NOCREATEROLE | Controls whether a user can create new users or roles using CREATE ROLE and CREATE USER . A user with the CREATEROLE permission can also modify and delete other users. | Default value:
NOCREATEROLE |

| Parameter | Description | Value Range/Default Value |
|--------------------------------|---|--|
| INHERIT
NOINHERIT | Controls whether a user can inherit the permissions of its group. You are not advised to use it. | - |
| AUDITADMIN
NOAUDITADMIN | Defines whether a user has the audit administrator attribute. | Default value:
NOAUDITADMIN |
| SYSADMIN
NOSYSADMIN | Controls whether a user is a system administrator. A user with the SYSADMIN attribute has the highest permission in the system. | Default value:
NOSYSADMIN |
| USEFT
NOUSEFT | Controls whether a user can perform operations on foreign tables, such as creating, deleting, modifying, and reading/writing foreign tables.

The default value is NOUSEFT , indicating that new users cannot perform operations on foreign tables by default.

The new user does not have the permission to perform operations on foreign tables. | - |
| LOGIN
NOLOGIN | Controls whether a user can log in to the database. Only users with the LOGIN attribute can log in to the database. | Default value: LOGIN |
| REPLICATION
NOREPLICATION | Controls whether a user is allowed to initiate streaming replication or put the system in and out of backup mode. A user with the REPLICATION attribute is only used for replication. | Default value:
NOREPLICATION |

| Parameter | Description | Value Range/Default Value |
|-----------------------------|--|---------------------------|
| INDEPENDENT NOINDEPENDENT | <p>Defines private and independent users. For a user with the INDEPENDENT attribute, administrators' rights to control and access this role are separated. Specific rules are as follows:</p> <ul style="list-style-type: none">Administrators have no rights to add, delete, query, modify, copy, or authorize the corresponding table objects without the authorization from the INDEPENDENT user.Without the authorization of the INDEPENDENT user, the administrator has no right to modify its inheritance relationship.The administrator does not have the permission to change the owner of the table object of an INDEPENDENT user.The administrator does not have the permission to remove the INDEPENDENT attribute of an INDEPENDENT user.The administrator does not have the permission to change the database password of an INDEPENDENT user. An INDEPENDENT must manage its own password. If the password is lost, it cannot be reset.The SYSADMIN attribute of a user cannot be changed to the INDEPENDENT attribute. | - |
| VCADMIN NOVCADMIN | <p>Defines a logical cluster administrator. A logical cluster administrator has the following more permissions than common users:</p> <ul style="list-style-type: none">Create, modify, and delete resource pools in the associated logical cluster.Grant the access permission for the associated logical cluster to other users or roles, or reclaim the access permission from those users or roles. | - |

| Parameter | Description | Value Range/Default Value |
|-------------------------|--|---|
| CONNECTION LIMIT | <p>Specifies the number of concurrent connections that can be used by a user on a single CN.</p> <p>To ensure that a cluster can run normally, the minimum value of CONNECTION LIMIT is the number of CNs in the cluster, because when you run ANALYZE on a CN of the cluster, other CNs will connect to the running CN for metadata synchronization. For example, if there are three CNs in a cluster, set CONNECTION LIMIT to 3 or a larger value.</p> | <p>Value range: integers no less than -1.</p> <p>Default value: -1, indicating there is no limit.</p> |
| ENCRYPTED UNENCRYPTED | <p>Determines whether the password stored in the system will be encrypted. (If neither is specified, the password status is determined by password_encryption_type.) According to product security requirements, the password must be stored encrypted. Therefore, UNENCRYPTED is forbidden in DWS. If the password is encrypted using SHA256, it will be stored as it is, regardless of whether it is specified as ENCRYPTED or UNENCRYPTED. This is because the system cannot decrypt the specified encrypted password. This allows reloading of the encrypted password during dump/restore.</p> <ul style="list-style-type: none">• password: login password.• DISABLE: Users can change their own passwords unless the passwords are disabled. You can use DISABLE to disable the password of a user. After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, IAM authentication, Kerberos authentication, or LDAP authentication. Only administrators can enable or disable passwords. Common users cannot disable the password of an initial user. To enable a password, run ALTER USER and specify the password. | <p>The password must:</p> <ul style="list-style-type: none">• Contain at least eight characters.• Differ from the username or it in reverse order.• Contain at least three types of the following characters: uppercase letters (A-Z), lowercase letters (a-z), digits (0-9), and special characters such as ~!@#\$%^&*()-_+=\ []{};,:<.>/? (If you use a character that is not in the specified range, an alarm is generated, but the character can still be created.) |

| Parameter | Description | Value Range/Default Value |
|-------------------------------|---|--|
| VALID BEGIN | Sets the timestamp when a user takes effect. If this clause is omitted, there is no restriction on when the user takes effect. | - |
| VALID UNTIL | Sets an expiration timestamp for an account. After the specified time, the account cannot be used. If this clause is omitted, it means the account will never expire | - |
| RESOURCE POOL | Sets the name of the resource pool used by a user. The name is from the system catalog pg_resource_pool . | - |
| USER GROUP 'groupuser' | Creates a subuser. | - |
| PERM SPACE | Sets the storage space of the user permanent table.
space_limit : specifies the upper limit of the storage space of the permanent table. | space_limit : A string consists of an integer and unit. The unit can be K/M/G/T/P.
0 indicates no limits. |
| TEMP SPACE | Sets the storage space of the user temporary table.
tmpspacelimit : specifies the storage space limit of the temporary table. | tmpspacelimit : A string consists of an integer and unit. The unit can be K/M/G/T/P.
0 indicates no limits. |
| SPILL SPACE | Sets the operator disk flushing space of the user.
spillspacelimit : specifies the operator spilling space limit. | spillspacelimit : A string consists of an integer and unit. The unit can be K/M/G/T/P.
0 indicates no limits. |
| NODE GROUP | Specifies the name of the logical cluster associated with a user. If the name contains uppercase characters or special characters, enclose the name with double quotation marks. | - |
| ACCOUNT LOCK ACCOUNT UNLOCK | Locks or unlocks an account. <ul style="list-style-type: none">• ACCOUNT LOCK: locks an account to forbid login to databases.• ACCOUNT UNLOCK: unlocks an account to allow login to databases. | - |

| Parameter | Description | Value Range/Default Value |
|------------------------|--|---------------------------|
| PGUSER | <p>This attribute is used to be compatible with open-source Postgres communication. An open-source Postgres client interface (Postgres 9.2.19 is recommended) can use a database user having this attribute to connect to the database.</p> <p>PGUSER of a user cannot be modified in the current version. For details, see PGUSER Parameter Setting Suggestions.</p> | - |
| AUTHINFO
'authinfo' | <p>This attribute is used to specify the user authentication type. authinfo is the description character string, which is case sensitive.</p> <p>Only the LDAP type is supported. Its description character string is ldap. LDAP authentication is an external authentication mode. Therefore, PASSWORD DISABLE must be specified.</p> <ul style="list-style-type: none">Additional information about LDAP authentication can be added to authinfo, for example, fulluser in LDAP authentication, which is equivalent to ldapprefix+username+ldapsuffix. If the content of authinfo is ldap, the user authentication type is LDAP. In this case, the ldapprefix and ldapsuffix information is provided by the corresponding record in the pg_hba.conf file.When executing ALTER ROLE, users are not allowed to change the authentication type. Only LDAP users are allowed to modify LDAP attributes. | - |

| Parameter | Description | Value Range/Default Value |
|-----------------------------------|--|---|
| PASSWORD EXPIRATION period | Number of days before the login password of the role expires. The user needs to change the password in time before the login password expires. If the login password expires, the user cannot log in to the system. In this case, the user needs to ask the administrator to set a new login password. | The value is an integer ranging from -1 to 999. <ul style="list-style-type: none">• Default value -1 indicates that there is no restriction.• 0 indicates that the login password expires immediately. |
| SET configuration_parameter | Sets the session parameters of a user. The session parameters modified by running the ALTER USER command apply only to the specified user and are valid in the next session started by the user. <ul style="list-style-type: none">• DEFAULT clears the value of configuration_parameter. The value of the configuration_parameter parameter inherits the default value of the new session of the role.• FROM CURRENT uses the value of configuration_parameter of the current session. | For details about the values of configuration_parameter and value , see SET . |
| RESET configuration_parameter/ALL | The effect of clearing the configuration_parameter value is the same as setting it to DEFAULT .
ALL indicates that all parameter values are cleared. | - |

PGUSER Parameter Setting Suggestions

The **PGUSER** attribute is for compatibility with the connection process. It does not handle incompatibilities due to kernel differences between the product and Postgres.

Users with the **PGUSER** attribute have a different authentication method. Open-source clients might reveal the **PGUSER** attribute through error messages. You are advised to use the client provided with the product. Example:

```
# normaluser is a user that does not have the PGUSER attribute. psql is the Postgres client tool.
pg@dws04:~> psql -d postgres -p 8000 -h 10.11.12.13 -U normaluser
psql: authentication method 10 not supported

# pguser is a user having the PGUSER attribute.
pg@dws04:~> psql -d postgres -p 8000 -h 10.11.12.13 -U pguser
Password for user pguser:
```

Examples

Create example user **u1**.

```
DROP USER IF EXISTS u1 CASCADE;  
CREATE USER u1 PASSWORD '{Password}';
```

Change the login password of user **u1**.

```
ALTER USER u1 IDENTIFIED BY '{new_Password}' REPLACE '{Password}';
```

Add the **CREATEROLE** permission to user **u1**.

```
ALTER USER u1 CREATEROLE;
```

Set the maximum number of database connections of user **u1** to **10**.

```
ALTER USER u1 with CONNECTION LIMIT 10;
```

Set the **enable_seqscan** parameter of the session associated with user **u1** to **on**. The setting takes effect in the next session.

```
ALTER USER u1 SET enable_seqscan TO on;
```

Reset the **enable_seqscan** parameter for user **u1**.

```
ALTER USER u1 RESET enable_seqscan;
```

Lock the **u1** account.

```
ALTER USER u1 ACCOUNT LOCK;
```

Links

[CREATE ROLE](#), [CREATE USER](#), and [DROP USER](#)

21.13.3.3 DROP USER

Function

Deleting a user will also delete the schema having the same name as the user.

Precautions

- **CASCADE** is used to delete objects (excluding databases) that depend on the user. **CASCADE** cannot delete locked objects unless the locked objects are unlocked or the processes that lock the objects are stopped.
- When deleting a user in the database, if the object that the user depends on is in another database or the object of the dependent user is another database, you need to manually delete the dependent objects in other databases or delete the dependent database. Then, delete the user. Cross-database cascading deletion cannot be performed.
- In a multi-tenant scenario, the service user will also be deleted when you delete a user group. If the specified **CASCADE** concatenation is deleted, **CASCADE** will be specified upon the deletion of the service user. If you fail to delete a user, an error is reported, and you cannot delete other users either.
- If the error table specified by the GDS foreign table created by user A is under the schema of user B, user B cannot be deleted by specifying the **CASCADE** keyword in **DROP USER**.

- If a "role is being used by other users" error is displayed when you run **DROP USER**, it might be that threads cannot respond to signals in a timely manner during the **CLEAN CONNECTION** process. As a result, connections are not completely cleared. In this case, you need to run **CLEAN CONNECTION** again.

⚠ DANGER

- Be cautious when using **DROP OBJECT** (e.g., **DATABASE, USER/ROLE, SCHEMA, TABLE, VIEW**) as it may cause data loss, especially with **CASCADE** deletions. Always back up data before proceeding.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

Syntax

```
DROP USER [ IF EXISTS ] user_name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

Table 21-132 DROP USER parameters

| Parameter | Description | Value Range |
|--------------------|---|--|
| IF EXISTS | Sends a notice instead of an error if the specified user does not exist. | - |
| user_name | Specifies the name of a user to be deleted. | Name of an existing user. |
| CASCADE RESTRICT | <p>Specifies how to process related data in the dependent object when a delete operation is performed.</p> <p>NOTE
In DWS, the postgresql.conf file contains the enable_kill_query parameter. This parameter affects the action of deleting user objects using CASCADE.</p> <ul style="list-style-type: none">• If enable_kill_query is on and CASCADE is used to delete user objects, the processes will be automatically killed and the user will be deleted at the same time.• If enable_kill_query is off and CASCADE is used to delete user objects, the user will be deleted after the processes are automatically killed. | <ul style="list-style-type: none">• CASCADE: automatically deletes all objects (such as tables) that depend on the user to be deleted. When a user is deleted in CASCADE mode, objects owner by the user and the user's permissions for objects will be deleted.• RESTRICT: refuses to delete the user if any objects depend on it. This is the default. |

Examples

Delete user **jim**.

```
DROP USER jim CASCADE;
```

Helpful Links

[ALTER USER](#), [CREATE USER](#)

21.13.3.4 CREATE ROLE

Function

Create a role.

A role is an entity that has own database objects and permissions. In different environments, a role can be considered a user, a group, or both.

Precautions

- **CREATE ROLE** adds a role to a database. The role does not have the login permission.
- Only the user who has the **CREATE ROLE** permission or a system administrator is allowed to create roles.

Syntax

```
CREATE ROLE role_name [ [ WITH ] option [ ... ] ] [ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' | DISABLE };
```

The syntax of role information configuration clause **option** is as follows:

```
{SYSADMIN | NOSYSADMIN}  
| {AUDITADMIN | NOAUDITADMIN}  
| {CREATEDB | NOCREATEDB}  
| {USEFT | NOUSEFT}  
| {CREATEROLE | NOCREATEROLE}  
| {INHERIT | NOINHERIT}  
| {LOGIN | NOLOGIN}  
| {REPLICATION | NOREPLICATION}  
| {INDEPENDENT | NOINDEPENDENT}  
| {VCADMIN | NOVCADMIN}  
| CONNECTION LIMIT connlimit  
| VALID BEGIN 'timestamp'  
| VALID UNTIL 'timestamp'  
| RESOURCE POOL 'respool'  
| USER GROUP 'groupuser'  
| PERM SPACE 'spacelimit'  
| TEMP SPACE 'tmpspacelimit'  
| SPILL SPACE 'spillspacelimit'  
| NODE GROUP logic_cluster_name  
| IN ROLE role_name [, ...]  
| IN GROUP role_name [, ...]  
| ROLE role_name [, ...]  
| ADMIN role_name [, ...]  
| USER role_name [, ...]  
| SYSID uid  
| DEFAULT TABLESPACE tablespace_name  
| PROFILE DEFAULT  
| PROFILE profile_name  
| PGUSER  
| AUTHINFO 'authinfo'  
| PASSWORD EXPIRATION period
```

Parameter Description

Table 21-133 CREATE ROLE parameters

| Parameter | Description | Value Range |
|-----------|---|--|
| role_name | Role name | A string. It must comply with the naming convention and can contain a maximum of 63 characters. For details, see Identifier Naming Conventions . |
| password | Specifies the login password. | <p>A password must:</p> <ul style="list-style-type: none">Contain at least eight characters. This is the default length.Differ from the user name or the user name spelled backwards.Contains at least three of the following four character types: uppercase letters, lowercase letters, digits, and special characters, including: ~!@#\$%^&*()-_+=+ [{ }];,;<.>/?. If you use characters other than the four types, a warning is displayed, but you can still create the password. <p>Value range: a string</p> |
| DISABLE | By default, you can change your password unless it is disabled. Use this parameter to disable the password of a user. | After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, IAM, Kerberos, LDAP, or oneaccess. Only administrators can enable or disable a password. Common users cannot disable the password of an initial user. To enable a password, run ALTER USER and specify the password. |

| Parameter | Description | Value Range |
|---------------------------|--|---|
| ENCRYPTED UNENCRYPTED | Determines whether the password stored in the system will be encrypted. | (If neither is specified, the password status is determined by password_encryption_type .) According to product security requirements, the password must be stored encrypted. Therefore, UNENCRYPTED is forbidden in DWS. If the password is encrypted using SHA256, it will be stored as it is, regardless of whether it is specified as ENCRYPTED or UNENCRYPTED . This is because the system cannot decrypt the specified encrypted password. This allows reloading of the encrypted password during dump/restore. |
| SYSADMIN NOSYSADMIN | Determines whether a new role is a system administrator. Roles having the SYSADMIN attribute have the highest permission. | The default value is NOSYSADMIN. |
| AUDITADMIN NOAUDITADMIN | Determines whether a role has the audit and management attributes. | The default value is NOAUDITADMIN. |
| CREATEDB NOCREATEDB | Defines a role's ability to create databases. A new role does not have the permission to create databases. | The default value is NOCREATEDB. |
| USEFT NOUSEFT | Determines whether a new role can perform operations on foreign tables, such as creating, deleting, modifying, and reading/writing foreign tables.

A new role does not have permissions for these operations. | The default value is NOUSEFT .
NOTE
In cluster 8.2.0 or later, if foreign_table_options of the GUC parameter security_enable_options is enabled, foreign table operations are permitted and the useft permission does not need to be granted to users. |

| Parameter | Description | Value Range |
|-----------------------------|--|-------------------------------------|
| CREATEROLE NOCREATEROLE | Determines whether a role will be permitted to create new roles (that is, execute CREATE ROLE and CREATE USER). A role with the CREATEROLE permission can also modify and delete other roles. | The default value is NOCREATEROLE. |
| INHERIT NOINHERIT | Determines whether a role "inherits" the permissions of roles it is a member of. You are not advised to execute them. | - |
| LOGIN NOLOGIN | Determines whether a role is allowed to log in to a database. A role having the LOGIN attribute can be thought of as a user. | The default value is NOLOGIN. |
| REPLICATION NOREPLICATION | Determines whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role having the REPLICATION attribute is a highly privileged role, and should only be used on roles used for replication. | The default value is NOREPLICATION. |

| Parameter | Description | Value Range |
|-----------------------------|--|---|
| INDEPENDENT NOINDEPENDENT | It defines a private and independent role. For a role with the INDEPENDENT attribute, administrators' rights to control and access this role are separated. | <p>The procedure is as follows:</p> <ul style="list-style-type: none">• Administrators have no rights to add, delete, query, modify, copy, or authorize the corresponding table objects without the authorization from the INDEPENDENT role.• Administrators have no rights to modify the inheritance relationship of the INDEPENDENT role without the authorization from this role.• Administrators have no rights to modify the owner of the table objects for the INDEPENDENT role.• Administrators have no rights to delete the INDEPENDENT attribute of the INDEPENDENT role.• Administrators have no rights to change the database password of the INDEPENDENT role. The INDEPENDENT role must manage its own password, which cannot be reset if lost.• The SYSADMIN attribute of a user cannot be changed to the INDEPENDENT attribute. |

| Parameter | Description | Value Range |
|------------------------|---|---|
| VCADMIN
NOVCADMIN | Defines the role of a logical cluster administrator. A logical cluster administrator has the following more permissions than common users: <ul style="list-style-type: none">• Create, modify, and delete resource pools in the associated logical cluster.• Grant the access permission for the associated logical cluster to other users or roles, or reclaim the access permission from those users or roles. | - |
| CONNECTION
LIMIT | Indicates how many concurrent connections the role can use on a single CN. | An integer that is greater than or equal to -1 . Default value: -1 , indicating there is no limit.
NOTICE
To ensure the proper running of a cluster, the minimum value of CONNECTION LIMIT is the number of CNs in the cluster, because when a cluster runs ANALYZE on a CN, other CNs will connect to the running CN for metadata synchronization. For example, if there are three CNs in the cluster, set CONNECTION LIMIT to 3 or a larger value. |
| VALID BEGIN | Sets a date and time when the role's password becomes valid. If this clause is omitted, the password will be valid for all time. | - |
| VALID UNTIL | Sets a date and time after which the role's password is no longer valid. If this clause is omitted, the password will be valid for all time. | - |

| Parameter | Description | Value Range |
|---------------------------|--|--|
| RESOURCE POOL | Sets the name of resource pool used by the role, and the name belongs to the system catalog:
pg_resource_pool. | - |
| USER GROUP
'groupuser' | Creates a sub-user. | - |
| PERM SPACE | Sets the storage space of the user permanent table. | space_limit: specifies the upper limit of the storage space of the permanent table. Value range: A string consists of an integer and unit. The unit can be K/M/G/T/P currently. 0 indicates no limits. |
| TEMP SPACE | Sets the storage space of the user temporary table. | tmpspacelimit: specifies the storage space limit of the temporary table. Value range: A string consists of an integer and unit. The unit can be K/M/G/T/P currently. 0 indicates no limits. |
| SPILL SPACE | Sets the operator disk flushing space of the user. | spillspacelimit: specifies the operator spilling space limit. Value range: A string consists of an integer and unit. The unit can be K/M/G/T/P currently. 0 indicates no limits. |
| NODE GROUP | Specifies the name of the logical cluster associated with a user. If the name contains uppercase characters or special characters, enclose the name with double quotation marks. | - |
| IN ROLE | Lists one or more existing roles whose permissions will be inherited by a new role. You are not advised to use them. | - |
| IN GROUP | Indicates an obsolete spelling of IN ROLE . You are not advised to use them. | - |

| Parameter | Description | Value Range |
|--------------------|--|---|
| ROLE | Lists one or more existing roles which are automatically added as members of the new role. | - |
| ADMIN | Is similar to ROLE . However, the roles after ADMIN can grant rights of new roles to other roles. | - |
| USER | Indicates an obsolete spelling of the ROLE clause. | - |
| SYSID | The SYSID clause is ignored. | - |
| DEFAULT TABLESPACE | The DEFAULT TABLESPACE clause is ignored. | - |
| PROFILE | The PROFILE clause is ignored. | - |
| PGUSER | This attribute is used to be compatible with open-source Postgres communication. An open-source Postgres client interface (Postgres 9.2.19 is recommended) can use a database user having this attribute to connect to the database. | <p>This attribute only ensures compatibility with the connection process. Incompatibility caused by kernel differences between this product and Postgres cannot be solved using this attribute.</p> <p>Users having the PGUSER attribute are authenticated in a way different from other users. Error information reported by the open-source client may cause the attribute to be enumerated. Therefore, you are advised to use a client of this product. Example:</p> <pre># normaluser is a user that does not have the PGUSER attribute. psql is the Postgres client tool. pg@dws04:~> psql -d postgres -p 8000 -h 10.11.12.13 -U normaluser psql: authentication method 10 not supported # pguser is a user having the PGUSER attribute. pg@dws04:~> psql -d postgres -p 8000 -h 10.11.12.13 -U pguser Password for user pguser:</pre> |

| Parameter | Description | Value Range |
|------------------------|---|---|
| AUTHINFO
'authinfo' | This attribute is used to specify the role authentication type. authinfo is the description character string, which is case sensitive. Currently, only the LDAP and OneAccess type is supported. Its description character string is ldap and oneaccess . LDAP and OneAccess are external authentication modes. Therefore, PASSWORD DISABLE must be specified. | <ul style="list-style-type: none">• Additional information about LDAP authentication can be added to authinfo, for example, fulluser in LDAP authentication, which is equivalent to ldapprefix+username+ldapsuffix. If the content of authinfo is ldap, the role authentication type is LDAP. In this case, the ldapprefix and ldapsuffix information is provided by the corresponding record in the pg_hba.conf file.• When the OneAccess authentication mode is supported, authinfo must contain oneaccessClientId and domain information in the 'oneaccessClientId=xxxx, domain=xxxx' format. oneaccessClientId indicates the ID of the oneaccess user and contains a maximum of 256 characters. domain indicates the access domain name of the oneaccess user and contains a maximum of 64 characters.• When executing the ALTER ROLE command, users are not allowed to change the authentication type. Only LDAP or OneAccess users are allowed to modify the LDAP or OneAccess attributes.• OneAccess is identified by internal code logic and does not need to be configured in pg_hba.conf. The authentication information is provided by authinfo. |

| Parameter | Description | Value Range |
|----------------------------|--|---|
| PASSWORD EXPIRATION period | Number of days before the login password of the role expires. The user needs to change the password in time before the login password expires. If the login password expires, the user cannot log in to the system. In this case, the user needs to ask the administrator to set a new login password. | An integer ranging from -1 to 999 . The default value is -1 , indicating that there is no restriction. The value 0 indicates that the login password expires immediately. |

Examples

Create the role **manager**.

```
CREATE ROLE manager IDENTIFIED BY '{password}';
```

Create a role with a validity period from January 1, 2015 to January 1, 2026.

```
CREATE ROLE miriam WITH LOGIN PASSWORD '{password}' VALID BEGIN '2015-01-01' VALID UNTIL '2026-01-01';
```

Create a role. The authentication type is LDAP. Other LDAP authentication information is provided by **pg_hba.conf**:

```
CREATE ROLE role1 WITH LOGIN AUTHINFO 'ldap' PASSWORD DISABLE;
```

Create a role. The authentication type is LDAP. The **fulluser** information for LDAP authentication is specified during the role creation. In this case, LDAP is case sensitive and must be enclosed in single quotation marks.

```
CREATE ROLE role2 WITH LOGIN AUTHINFO 'ldapcn=role2,cn=user,dc=lework,dc=com' PASSWORD DISABLE;
```

Create a role and set the authentication type to **OneAccess**. The authinfo information of OneAccess authentication is specified during the role creation. In this case, **oneaccess** is case sensitive and must be enclosed in single quotation marks.

```
CREATE ROLE role3 WITH LOGIN AUTHINFO 'oneaccessClientId=AbCd123,domain=example.exampleoneaccess.com' PASSWORD DISABLE;
```

Create a role and set the validity period of its login password to 30 days.

```
CREATE ROLE role4 WITH LOGIN PASSWORD '{password}' PASSWORD EXPIRATION 30;
```

Helpful Links

[SET ROLE](#), [ALTER ROLE](#), [DROP ROLE](#), [GRANT](#), [REVOKE](#)

21.13.3.5 ALTER ROLE

Function

ALTER ROLE changes the attributes of a role.

Precautions

None

Syntax

- Modifying the Rights of a Role

```
ALTER ROLE role_name [ [ WITH ] option [ ... ] ];
```

The **option** clause for granting rights is as follows:

```
{CREATEDB | NOCREATEDB}  
| {CREATEROLE | NOCREATEROLE}  
| {INHERIT | NOINHERIT}  
| {AUDITADMIN | NOAUDITADMIN}  
| {SYSADMIN | NOSYSADMIN}  
| {USEFT | NOUSEFT}  
| {LOGIN | NOLOGIN}  
| {REPLICATION | NOREPLICATION}  
| {INDEPENDENT | NOINDEPENDENT}  
| {VCADMIN | NOVCADMIN}  
| CONNECTION LIMIT connlimit  
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'  
| [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY 'password' [ REPLACE 'old_password' ]  
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD { 'password' | DISABLE }  
| [ ENCRYPTED | UNENCRYPTED ] IDENTIFIED BY { 'password' [ REPLACE 'old_password' ] |  
DISABLE }  
| VALID BEGIN 'timestamp'  
| VALID UNTIL 'timestamp'  
| RESOURCE POOL 'respool'  
| USER GROUP 'groupuser'  
| PERM SPACE 'spacelimit'  
| TEMP SPACE 'tmpspacelimit'  
| SPILL SPACE 'spillspacelimit'  
| NODE GROUP logic_cluster_name  
| ACCOUNT { LOCK | UNLOCK }  
| PGUSER  
| AUTHINFO 'authinfo'  
| PASSWORD EXPIRATION period
```

- Rename a role.

```
ALTER ROLE role_name  
  RENAME TO new_name;
```

- Set parameters for a role.

```
ALTER ROLE role_name [ IN DATABASE database_name ]  
  SET configuration_parameter [{ TO | = } { value | DEFAULT } | FROM CURRENT];
```

- Reset parameters for a role.

```
ALTER ROLE role_name  
  [ IN DATABASE database_name ] RESET {configuration_parameter|ALL};
```

Parameter Description

Table 21-134 ALTER ROLE parameters

| Parameter | Description | Value Range |
|---------------------------|---|---|
| role_name | Specifies the name of the role to be modified. | Name of an existing role. |
| new_name | Indicates the new name of a role. | A string compliant with the identifier naming rules . |
| CREATEDB NOCREATEDB | Defines a role's ability to create databases. If not specified, NOCREATEDB is the default value.
A new role does not have the permission to create databases. | - |
| CREATEROLE NOCREATEROLE | Determines whether a role will be permitted to create new roles (that is, execute CREATE ROLE and CREATE USER). If not specified, NOCREATEROLE is the default value.
A role with the CREATEROLE permission can also modify and delete other roles. | - |
| INHERIT NOINHERIT | Determines whether the role can inherit permissions of its group. You are not advised to execute them. | - |
| AUDITADMIN NOAUDITADMIN | Determines whether a role has the audit and management attributes.
If not specified, NOAUDITADMIN is the default. | - |
| SYSADMIN NOSYSADMIN | Determines whether a new role is a system administrator. Roles having the SYSADMIN attribute have the highest permission.
If not specified, NOSYSADMIN is the default value. | - |
| USEFT NOUSEFT | Determines whether a new role can perform operations on foreign tables, such as creating, deleting, modifying, and reading/writing foreign tables.
A new role does not have permissions for these operations.
Default value NOUSEFT indicates that a new role cannot perform operations on foreign tables. | - |

| Parameter | Description | Value Range |
|-----------------------------|--|-------------|
| LOGIN NOLOGIN | Indicates whether a user can log in to the database. Only roles with the LOGIN attribute can log in to the database. A role having the LOGIN attribute can be thought of as a user.
Value range: If not specified, NOLOGIN is the default. | - |
| REPLICATION NOREPLICATION | Determines whether a role is allowed to initiate streaming replication or put the system in and out of backup mode. A role having the REPLICATION attribute is a highly privileged role, and should only be used on roles used for replication.
If not specified, NOREPLICATION is the default. | - |

| Parameter | Description | Value Range |
|-----------------------------|---|-------------|
| INDEPENDENT NOINDEPENDENT | <p>Defines private and independent roles. For a role with the INDEPENDENT attribute, administrators' rights to control and access this role are separated. Specific rules are as follows:</p> <ul style="list-style-type: none">• Administrators have no rights to add, delete, query, modify, copy, or authorize the corresponding table objects without the authorization from the INDEPENDENT role.• Administrators have no rights to modify the inheritance relationship of the INDEPENDENT role without the authorization from this role.• Administrators have no rights to modify the owner of the table objects for the INDEPENDENT role.• Administrators have no rights to delete the INDEPENDENT attribute of the INDEPENDENT role.• Administrators have no rights to change the database password of the INDEPENDENT role. The INDEPENDENT role must manage its own password, which cannot be reset if lost.• The SYSADMIN attribute of a user cannot be changed to the INDEPENDENT attribute. | - |
| VCADMIN NOVCADMIN | <p>Defines the role of a logical cluster administrator. A logical cluster administrator has the following more permissions than common users:</p> <ul style="list-style-type: none">• Create, modify, and delete resource pools in the associated logical cluster.• Grant the access permission for the associated logical cluster to other users or roles, or reclaim the access permission from those users or roles. | - |

| Parameter | Description | Value Range |
|---------------------|---|--|
| CONNECTION
LIMIT | <p>Indicates the number of concurrent connections that a role can use on a single CN.</p> <p>To ensure the proper running of a cluster, the minimum value of CONNECTION LIMIT is the number of CNs in the cluster, because when a cluster runs ANALYZE on a CN, other CNs will connect to the running CN for metadata synchronization. For example, if there are three CNs in the cluster, set CONNECTION LIMIT to 3 or a larger value.</p> | <p>Integers no less than -1.</p> <p>The default value -1 means no limit.</p> |

| Parameter | Description | Value Range |
|-------------------------|---|--|
| ENCRYPTED UNENCRYPTED | <p>Determines whether the password stored in the system will be encrypted. (If neither is specified, the password status is determined by password_encryption_type.) According to product security requirements, the password must be stored encrypted. Therefore, UNENCRYPTED is forbidden in DWS. If the password is encrypted using SHA256, it will be stored as it is, regardless of whether it is specified as ENCRYPTED or UNENCRYPTED. This is because the system cannot decrypt the specified encrypted password. This allows reloading of the encrypted password during dump/restore.</p> <ul style="list-style-type: none">password
Specifies the login password.DISABLE
By default, you can change your password unless it is disabled. Use this parameter to disable the password of a user. After the password of a user is disabled, the password will be deleted from the system. The user can connect to the database only through external authentication, for example, IAM authentication, Kerberos authentication, or LDAP authentication. Only administrators can enable or disable a password. Common users cannot disable the password of an initial user. To enable a password, run ALTER USER and specify the password. | <p>The password must:</p> <ul style="list-style-type: none">Contain at least eight characters.Differ from the username or it in reverse order.Contain at least three types of the following characters:
uppercase letters (A-Z), lowercase letters (a-z), digits (0-9), and special characters such as ~!@#\$%^&*()-_+=+\\[{}];,;<.>/? (If you use a character that is not in the specified range, an alarm is generated, but the character can still be created.) |
| VALID BEGIN | Sets a date and time when the role's password becomes valid. If this clause is omitted, the password will be valid for all time. | - |
| VALID UNTIL | Sets a date and time after which the role's password is no longer valid. If this clause is omitted, the password will be valid for all time. | - |

| Parameter | Description | Value Range |
|-------------------------------------|--|---|
| RESOURCE POOL | Sets the name of resource pool used by a role, and the name belongs to the system catalog:
pg_resource_pool . | - |
| USER GROUP
'groupuser' | Creates a sub-user. | - |
| PERM SPACE | Sets the storage space of the user permanent table.
space_limit specifies the upper limit of the storage space of the permanent table. | space_limit : A string consists of an integer and unit. The unit can be K/M/G/T/P. 0 indicates no limits. |
| TEMP SPACE | Sets the storage space of the user temporary table.
tmpspacelimit specifies the storage space limit of the temporary table. | tmpspacelimit : A string consists of an integer and unit. The unit can be K/M/G/T/P. 0 indicates no limits. |
| SPILL SPACE | Sets the operator disk flushing space of the user.
spillspacelimit specifies the operator spilling space limit. | spillspacelimit : A string consists of an integer and unit. The unit can be K/M/G/T/P. 0 indicates no limits. |
| NODE GROUP | Specifies the name of the logical cluster associated with a user. If the name contains uppercase characters or special characters, enclose the name with double quotation marks. | - |
| ACCOUNT LOCK
ACCOUNT
UNLOCK | <ul style="list-style-type: none">● ACCOUNT LOCK: locks an account to forbid login to databases.● ACCOUNT UNLOCK: unlocks an account to allow login to databases. | - |

| Parameter | Description | Value Range |
|----------------------------------|---|---|
| PGUSER | <p>This attribute is used to be compatible with open-source Postgres communication. An open-source Postgres client interface (Postgres 9.2.19 is recommended) can use a database user having this attribute to connect to the database.</p> <p>PGUSER of a role cannot be modified in the current version. For details, see PGUSER Parameter Setting Suggestions.</p> | - |
| AUTHINFO
'authinfo' | <p>Specifies the role authentication type. authinfo is the description character string, which is case sensitive. Only the LDAP type is supported. Its description character string is ldap. LDAP authentication is an external authentication mode. Therefore, PASSWORD DISABLE must be specified.</p> <ul style="list-style-type: none">• Additional information about LDAP authentication can be added to authinfo, for example, fulluser in LDAP authentication, which is equivalent to ldapprefix+username+ldapsuffix. If the content of authinfo is ldap, the role authentication type is LDAP. In this case, the ldapprefix and ldapsuffix information is provided by the corresponding record in the pg_hba.conf file.• When executing the ALTER ROLE command, users are not allowed to change the authentication type. Only LDAP users are allowed to modify LDAP attributes. | - |
| PASSWORD
EXPIRATION
period | <p>Specifies the number of days before the login password of the role expires. You need to change the password before the password expires. If your login password expires, you cannot log in. Ask the administrator for a new password.</p> | <p>An integer ranging from -1 to 999.</p> <ul style="list-style-type: none">• Default value -1 indicates that there is no restriction.• 0 indicates that the login password expires immediately. |

| Parameter | Description | Value Range |
|--------------------------------------|--|--|
| IN DATABASE
database_name | Modifies the parameters of a role in a specified database. | - |
| SET
configuration_parameter | Sets parameters for a role. The session parameters modified using the ALTER ROLE command is only for a specific role and is valid in the next session triggered by the role. <ul style="list-style-type: none">• DEFAULT clears the value of configuration_parameter. The value of the configuration_parameter parameter inherits the default value of the new session of the role.• FROM CURRENT uses the value of configuration_parameter of the current session. | Values of configuration_parameter and value are listed in SET . |
| RESET
configuration_parameter/ALL | The effect of clearing the configuration_parameter value is the same as setting it to DEFAULT . ALL indicates that all parameter values are cleared. | - |

PGUSER Parameter Setting Suggestions

The **PGUSER** attribute is for compatibility with the connection process. It does not handle incompatibilities due to kernel differences between the product and Postgres.

Users having the **PGUSER** attribute are authenticated in a way different from other users. Error information reported by the open-source client may cause the attribute to be enumerated. Therefore, you are advised to use a client of this product. Example:

```
# normaluser is a user that does not have the PGUSER attribute. psql is the Postgres client tool.
pg@dws04:~> psql -d postgres -p 8000 -h 10.11.12.13 -U normaluser
psql: authentication method 10 not supported

# pguser is a user having the PGUSER attribute.
pg@dws04:~> psql -d postgres -p 8000 -h 10.11.12.13 -U pguser
Password for user pguser:
```

Examples

Create example roles **r1**, **r2**, and **r3**.

```
CREATE ROLE r1 IDENTIFIED BY '{Password}';
CREATE ROLE r2 WITH LOGIN AUTHINFO 'ldapcn=r2,cn=user,dc=lework,dc=com' PASSWORD DISABLE;
CREATE ROLE r3 WITH LOGIN PASSWORD '{Password}' PASSWORD EXPIRATION 30;
```

Modify the login permission of role **r1**.

```
ALTER ROLE r1 login;
```

Change the password of role **r1**.

```
ALTER ROLE r1 IDENTIFIED BY '{new_Password}' REPLACE '{Password}';
```

Change the **manager** role to the system administrator.

```
ALTER ROLE r1 SYSADMIN;
```

Modify the **fulluser** information of the LDAP authentication role.

```
ALTER ROLE r2 WITH LOGIN AUTHINFO 'ldapcn=role2,cn=user2,dc=func,dc=com' PASSWORD DISABLE;
```

Change the validity period of the login password of the role to 90 days.

```
ALTER ROLE r3 PASSWORD EXPIRATION 90;
```

Links

[CREATE ROLE](#), [DROP ROLE](#), and [SET](#).

21.13.3.6 SET ROLE

Function

SET ROLE sets the current user identifier of the current session.

Precautions

- Users of the current session must be members of specified **rolename**, but the system administrator can choose any roles.
- Executing this command may add rights of a user or restrict rights of a user. If the role of a session user has the **INHERITS** attribute, it automatically has all rights of roles that **SET ROLE** enables the role to be. In this case, **SET ROLE** physically deletes all rights directly granted to session users and rights of its belonging roles and only leaves rights of the specified roles. If the role of the session user has the **NOINHERITS** attribute, **SET ROLE** deletes rights directly granted to the session user and obtains rights of the specified role.

Syntax

- Set the current user identifier of the current session.

```
SET [ SESSION | LOCAL ] ROLE role_name PASSWORD '{password}';
```
- Reset the current user identifier to that of the current session.

```
RESET ROLE;
```

Parameter Description

Table 21-135 SET ROLE parameters

| Parameter | Description | Value Range |
|------------|--|---|
| SESSION | Specifies that the command takes effect only for the current session. This parameter is used by default. | The value is a string and must comply with the rules in Identifier Naming Conventions . |
| LOCALE | Indicates that the specified command takes effect only for the current transaction. | - |
| role_name | Specifies the role name. | The value is a string and must comply with the rules in Identifier Naming Conventions . |
| password | Specifies the password of a role. It must comply with the password convention. | - |
| RESET ROLE | Resets the current user identifier. | - |

Examples

Create the role **manager**.

```
CREATE ROLE paul IDENTIFIED BY '{Password}';
```

Set the current user to **paul**.

```
SET ROLE paul PASSWORD '{password}';
```

View the current session user and the current user.

```
SELECT SESSION_USER, CURRENT_USER;
```

Reset the current user.

```
RESET role;
```

21.13.3.7 DROP ROLE

Function

DROP ROLE deletes a specified role.

Precautions

If a "role is being used by other users" error is displayed when you run **DROP ROLE**, it might be that threads cannot respond to signals in a timely manner

during the **CLEAN CONNECTION** process. As a result, connections are not completely cleared. In this case, you need to run **CLEAN CONNECTION** again.

** DANGER**

- Be cautious when using **DROP OBJECT** (e.g., **DATABASE, USER/ROLE, SCHEMA, TABLE, VIEW**) as it may cause data loss, especially with **CASCADE** deletions. Always back up data before proceeding.
 - For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.
-

Syntax

```
DROP ROLE [ IF EXISTS ] role_name [, ...];
```

Parameter Description

Table 21-136 DROP ROLE parameters

| Parameter | Description | Value Range |
|-----------|--|-------------------|
| IF EXISTS | Sends a notice instead of an error if the specified role does not exist. | - |
| role_name | Specifies the name of the role to be deleted. | An existing role. |

Examples

Delete **manager** role.

```
DROP ROLE manager;
```

Helpful Links

[CREATE ROLE](#), [ALTER ROLE](#), [SET ROLE](#)

21.13.3.8 CREATE GROUP

Function

CREATE GROUP creates a user group.

Precautions

CREATE GROUP is an alias for **CREATE ROLE**, and it is not a standard SQL command and not recommended. Users can use **CREATE ROLE** directly.

Syntax

```
CREATE GROUP group_name [ [ WITH ] option [ ... ] ]  
[ ENCRYPTED | UNENCRYPTED ] { PASSWORD | IDENTIFIED BY } { 'password' | DISABLE };
```

The syntax of optional **action** clause is as follows:

```
where option can be:  
{SYSADMIN | NOSYSADMIN}  
| {AUDITADMIN | NOAUDITADMIN}  
| {CREATEDB | NOCREATEDB}  
| {USEFT | NOUSEFT}  
| {CREATEROLE | NOCREATEROLE}  
| {INHERIT | NOINHERIT}  
| {LOGIN | NOLOGIN}  
| {REPLICATION | NOREPLICATION}  
| {INDEPENDENT | NOINDEPENDENT}  
| {VCADMIN | NOVADMIN}  
| CONNECTION LIMIT connlimit  
| VALID BEGIN 'timestamp'  
| VALID UNTIL 'timestamp'  
| RESOURCE POOL 'respool'  
| USER GROUP 'groupuser'  
| PERM SPACE 'spacelimit'  
| NODE GROUP logic_group_name  
| IN ROLE role_name [, ...]  
| IN GROUP role_name [, ...]  
| ROLE role_name [, ...]  
| ADMIN role_name [, ...]  
| USER role_name [, ...]  
| SYSID uid  
| DEFAULT TABLESPACE tablespace_name  
| PROFILE DEFAULT  
| PROFILE profile_name  
| PGUSER
```

Parameter Description

See [Parameter Description](#) in **CREATE ROLE**.

Helpful Links

[ALTER GROUP](#), [DROP GROUP](#), and [CREATE ROLE](#)

21.13.3.9 ALTER GROUP

Function

This syntax modifies the attributes of a user group.

Precautions

ALTER GROUP is an alias for **ALTER ROLE**, and it is not a standard SQL command and not recommended. Users can use **ALTER ROLE** directly.

Syntax

- Add users to a group.

```
ALTER GROUP group_name  
ADD USER user_name [, ... ];
```
- Remove users from a group.


```
ALTER GROUP group_name  
DROP USER user_name [, ... ];
```

- Change the name of the group.

```
ALTER GROUP group_name  
RENAME TO new_name;
```

Parameter Description

For details about more parameters, see [Parameter Description](#) in **ALTER ROLE**.

Table 21-137 ALTER GROUP parameters

| Parameter | Description | Value Range |
|------------|--|---|
| group_name | Specifies the name of the user group to be modified. | A string that indicates the name of an existing group. |
| user_name | Specifies the user to be added to or removed from the group. | Name of an existing user. |
| new_name | Specifies the new name of a group. | A string compliant with the identifier naming rules . |

Helpful Links

[CREATE GROUP](#), [DROP GROUP](#), and [ALTER ROLE](#)

21.13.3.10 DROP GROUP

Function

DROP GROUP deletes a user group.

DROP GROUP is the alias for **DROP ROLE**.

Precautions

DROP GROUP is the internal interface encapsulated in the **gs_om** tool. You are not advised to use this interface, because doing so affects the cluster.

Syntax

```
DROP GROUP [ IF EXISTS ] group_name [, ...];
```

Parameter Description

See [Parameter Description](#) in **DROP ROLE**.

Helpful Links

[CREATE GROUP](#), [ALTER GROUP](#), [DROP ROLE](#)

21.13.3.11 DROP OWNED

Function

DROP OWNED deletes the database objects of a database role.

Precautions

The role's permissions on all the database objects in the current database and shared objects (databases and tablespaces) are revoked.

Syntax

```
DROP OWNED BY name [, ...] [ CASCADE | RESTRICT ];
```

Parameter Description

Table 21-138 DROP OWNED parameters

| Parameter | Description | Value Range |
|--------------------|---|---|
| name | Name of the role whose objects are to be deleted and whose permissions are to be revoked. | An existing role name. |
| CASCADE RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none">• CASCADE: automatically deletes objects that depend on the affected objects.• RESTRICT (default): refuses to delete objects with dependent objects. |

Example

Remove all database objects owned by role **u1**.

```
DROP OWNED BY u1;
```

21.13.4 Tables and External Data Sources

21.13.4.1 CREATE TABLE

Function

Creates a new empty table in the current database.

This table is owned by the user who executes the command. However, if the system administrator creates a table in the schema with the same name as a common user, the owner of the table is the user (not the system administrator).

Precautions

Table 21-139 Precautions for CREATE TABLE

| Constraint | Description |
|--|---|
| Number of column-store and HDFS partitioned tables that can be created | For better performance, keep this value at or below 1,000. |
| Data types supported by column-store tables | For details, see Data Types Supported by Column-Store Tables . |
| Primary key and unique constraint requirements | The primary key and unique constraints in a table must contain distribution columns. |
| Modification of distribution columns | The data type of the distribution column in an existing table cannot be modified. |
| Row-store REPLICATION tables | A system column cannot be designated as a primary key in a row-store REPLICATION distributed table. |
| Supported range of column-store tables | Column-store tables support the PARTIAL CLUSTER KEY and table-level primary key and unique constraints, but do not support table-level foreign key constraints. |
| Columns of column-store tables | Only the NULL , NOT NULL , and DEFAULT constant values are supported. |
| Hot and cold tables | <ul style="list-style-type: none">Hot and cold tables support only column-store partitioned tables and depend on the available OBS.Hot and cold tables support only the default tablespace default_obs_tbs. If you need to add an OBS tablespace, contact technical support. |
| Delta tables supported by column-store tables | Whether column-store tables support a delta table is specified by the enable_delta parameter. The threshold for storing data into a delta table is specified by the deltarow_threshold parameter. Column-store tables with delta tables are not recommended. If delta tables are used, disk bloat and performance deterioration may occur due to insufficient merge time. Delta tables in column-store tables are no longer supported in versions later than 9.1.0.200 due to the preceding problems. HStore tables are used instead. For details about HStore tables, see "Hybrid Data Warehouse" in <i>Data Warehouse Service (DWS) Developer Guide</i> . |

| Constraint | Description |
|---------------|--|
| HStore tables | <ul style="list-style-type: none">When creating an HStore table, ensure that the database GUC parameter settings meet the following requirements:<ul style="list-style-type: none">autovacuum is set to on.The value of autovacuum_max_workers_hstore is greater than 0.The value of autovacuum_max_workers is greater than that of autovacuum_max_workers_hstore. |
| V3 tables | <ul style="list-style-type: none">Once a table is created, it cannot be switched from a non-V3 table to a V3 table using the ALTER TABLE syntax. In other words, a non-V3 table (colversion is 2.0) cannot be converted to a V3 table (colversion is 3.0).Delta tables and column-store level-2 partitions cannot be set for V3 tables (colversion is 3.0, storage-compute decoupling).V3 tables cannot be configured as HStore tables, cold and hot tables, or time sequence tables.Global temporary tables and temporary tables cannot be created for V3 tables. Any created temporary tables are automatically converted to temporary tables with colversion 2.0. |
| Other | If an error occurs during table creation, the system may not be able to clear the created, empty disk file automatically after the system is recovered. This problem rarely occurs and does not affect the normal running of the database system. |

WARNING

- Do not specify a custom tablespace when creating an ordinary table.
- Do not specify the **COMPRESS** compression attribute when creating a row-store table.
- When creating a hash-distributed table object, ensure that data is evenly distributed. (For a table with more than 10 GB data, the skew rate must be less than 10%.)
- When creating a replication-distributed table object, ensure that there are less than 1 million rows in a table.
- For a large table (with more than 50 million rows of data) that contains the time field, ensure that the table is designed as a partitioned table and the partition interval is properly designed based on the query characteristics.
- For a table that has a large amount of data to be added, deleted, or modified, keep the number of indexes within 3 (maximum value: 5).
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

Syntax

```
CREATE [ [ GLOBAL | LOCAL | VOLATILE ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name
{ ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
  | table_constraint
  | LIKE source_table [ like_option [...] ] }
  [, ... ] )
  LIKE source_table [ like_option [...] ] }
[ WITH ( { storage_parameter = value } [, ... ] ) ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS } ]
[ COMPRESS | NOCOMPRESS ]
[ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [...] ) } } ]
[ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
[ COMMENT [=] 'text' ];
```

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  ON UPDATE on_update_expr |
  COMMENT 'text' |
  UNIQUE [ NULLS [NOT] DISTINCT | NULLS IGNORE ] index_parameters |
  PRIMARY KEY index_parameters |
  REFERENCES reftable [ ( refcolumn ) ] }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **compress_mode** of a column is as follows:

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

- **table_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE [ NULLS [NOT] DISTINCT | NULLS IGNORE ] ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  PARTIAL CLUSTER KEY ( column_name [, ... ] ) }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

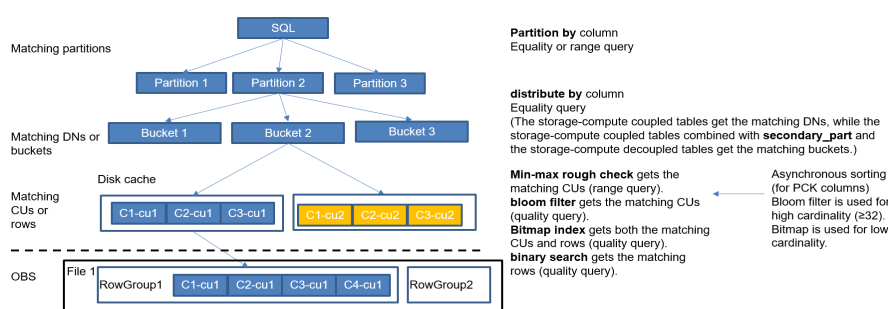
- **like_option** is as follows:

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS |  
PARTITION | REOPTIONS | DISTRIBUTION | DROPCOLUMNS | ALL }
```

- **index_parameters** is as follows:
[WITH ({storage_parameter = value} [, ...])]

Table Design Reference

DWS is compatible with the PostgreSQL ecosystem. Row storage and its B-tree index are similar to those of PostgreSQL. Column storage and its index are self-developed. When creating a table, it is crucial to choose the right storage method, distribution column, partition key, and index. This ensures efficient data access during SQL execution, reducing I/O consumption. The following figure illustrates the process from SQL statement initiation to data acquisition, helping you understand the function of each technical method for performance optimization.



1. When the SQL statement is executed, the partitioned table is optimized using the Partition Column to pinpoint the specific partition.
2. The Distribute Column is used in a distributed hash table to quickly identify the data shard where the data resides. The data shard is located on a DN in a storage-compute coupled architecture, while in a storage-compute decoupled architecture, it is located on a bucket.
3. In row-store mode, the B-tree index is used to quickly locate the data page. In column-store mode, the **min-max** index is used to quickly locate the CU data block that may contain relevant data. This index is particularly effective when filtering on the PCK column.
4. The system automatically maintains the **min-max** index for all columns in the column-store mode. There is no need for manual index definition. The **min-max** index is used for coarse filtering. CU data blocks meeting the min-max condition may not contain data rows that meet the filter condition. If a bitmap column is defined, the bitmap index can quickly locate the row number of data that meets the filter condition in the CU. For ordered CUs, binary search is also used to quickly locate the row number of data.
5. Column storage supports B-tree and GIN indexes, which can quickly locate the CU and row number of data that meets the conditions. However, due to high index maintenance costs, you are advised to use bitmap indexes instead, unless there are high performance requirements for point queries.

Table 21-140 lists the existing optimization methods of DWS.

Table 21-140 Optimization methods

| No. | Method | Usage | Example SQL | Modifiable After Creation |
|-----|-----------------------|--|---|--|
| 1 | String | <ol style="list-style-type: none">1. The string type has slower performance compared to the fixed-length type, so it is not recommended for scenarios where the fixed-length type is more suitable.2. If the specified length is less than 16, performance will be significantly improved. | - | Yes (The existing data can be rewritten.) |
| 2 | Numeric | Specifying precision for the numeric type is essential for improving performance. It is not advisable to use the numeric type without specifying precision. | - | Yes (The existing data can be rewritten.) |
| 3 | Partition by Column | <ol style="list-style-type: none">1. This requires user-defined settings and is designed for partitioned tables. Pruning is possible using partition keys and partition-wise joins are supported. This method is suitable for equality and range queries.2. Having more than 1000 partitions is not recommended, and it is advisable to limit the number of partition columns to two. | <pre>SELECT * FROM t1 WHERE t1.c1='p1';</pre> | No (You need to create a new table to make modifications.) |
| 4 | secondary_part_column | <ol style="list-style-type: none">1. This requires user-defined settings and is applicable only to column-store tables and equality queries.2. Specify a level-2 partition on the most commonly used equivalent filter. | <pre>SELECT * FROM t1 WHERE t1.c1='p1';</pre> | No (You need to create a new table to make modifications.) |

| No. | Method | Usage | Example SQL | Modifiable After Creation |
|-----|----------------------|--|--|---|
| 5 | Distribute by Column | This requires user-defined settings and is suitable for join fields that require frequent GROUP BY or multi-table joins. It reduces data shuffling through local joins and is ideal for equality queries. | <pre>SELECT * FROM t1 join t2 on t1.c3 = t2.c1;</pre> | No (You need to create a new table to make modifications.) |
| 6 | Bitmap column | Define the bitmap index (cardinality ≤ 32) or bloom filter (cardinality > 32) based on the repeated values in the CU. This method is applicable to equivalent queries of varchar or text type columns. It is advised to create indexes on columns involved in the WHERE condition. | <pre>SELECT * FROM t1 WHERE t1.c4='hello';</pre> | Yes (Modification does not rewrite existing data. Only the new data is affected.) |
| 7 | min-max index | <ol style="list-style-type: none">1. The min-max index is automatically generated and can be used for both equality and range queries.2. The min-max filtering effect depends on the data order. Specifying the PCK column enhances the filtering effect. | <pre>SELECT * FROM t1 WHERE c3 > 100 and c3 < 200;</pre> | Yes (The PCK columns can be modified. Modification does not rewrite existing data and only the new data is affected.) |

| No. | Method | Usage | Example SQL | Modifiable After Creation |
|-----|----------------------------|---|---|--|
| 8 | Primary key (B-tree index) | <ol style="list-style-type: none">1. UPSERT data import strongly depends on the primary key and needs to be customized. It is applicable to equality and range queries. We suggest limiting the number of columns to five or fewer.2. If service requirements are met, it is better to use fixed-length type columns. During definition, place columns with more distinct values at the beginning. | <pre>SELECT * FROM t1 WHERE c3 > 100 and c3 < 200;</pre> | Yes (The index can be modified and re-created.) |
| 9 | GIN index | <ol style="list-style-type: none">1. This requires user-defined settings and is suitable for multi-condition equality queries. Avoid using columns with more than 1 million distinct values.2. It is recommended when the data volume after filtering is less than 1000. If the data volume remains large after filtering, it is not recommended. | <pre>SELECT * FROM t1 WHERE c1 = 100 and c3 = 200 and c2 = 105;</pre> | Yes (The index can be modified and re-created.) |
| 10 | Orientation=column/row | This method specifies whether a table is stored in rows or columns. Row-store tables cannot be compressed and are best suited for point queries and frequent updates. Column-store tables can be compressed and are ideal for analysis purposes. | - | No (You need to create a new table to make modifications.) |

Parameter Description

Table 21-141 CREATE TABLE parameters

| Parameter | Description | Value Range or Example |
|-----------|---|--|
| UNLOGGED | If UNLOGGED is specified, the created table is an unlogged table. Data written to unlogged tables is not written to the write-ahead log, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after a crash or unclean shutdown, incurring data loss risks. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are not automatically logged as well. | <p>Usage scenario: Unlogged tables do not ensure safe data. You can back up data before using unlogged tables; for example, you should back up the data before a system upgrade.</p> <p>Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, you should re-create the indexes with errors.</p> <p>Example:</p> <pre>CREATE UNLOGGED TABLE table_unlog (col1 VARCHAR(50) , col2 TEXT);</pre> |

| Parameter | Description | Value Range or Example |
|-----------|--|------------------------|
| | <p>NOTICE</p> <ol style="list-style-type: none">1. The UNLOGGED table uses no primary/standby mechanism. In the case of system faults or abnormal breakpoints, data loss may occur. Therefore, the UNLOGGED table cannot be used to store basic data.2. Starting from version 9.1.0, UNLOGGED tables are automatically saved in the pg_unlogged tablespace and cannot be moved or assigned to other tablespaces.3. After an earlier version is upgraded to 9.1.0, the UNLOGGED table created in the earlier version is still stored in the original tablespace.4. If the instance restarts unexpectedly, the UNLOGGED table will be reset, which can impact the instance's recovery time objective (RTO). In version 9.1.0, switch_unlogged_tablespace.py can be used to move unlogged tables to optimize the recovery time objective (RTO). This script works together with the GUC parameter enable_unlogged_tablespace_compat. | |

| Parameter | Description | Value Range or Example |
|---------------------------|--|--|
| GLOBAL LOCAL VOLATILE | <p>Specify the keywords GLOBAL, LOCAL, and VOLATILE before TEMP or TEMPORARY to create temporary tables with different attributes. Global temporary tables are supported only by 8.2.1.220 and later cluster versions.</p> <ul style="list-style-type: none">• If LOCAL is specified, a local temporary table is created.• If keyword GLOBAL is specified, the attributes of the temporary table depend on the GUC parameter enable_global_temp_table. The default value of this parameter is ON.
If enable_global_temp_table is set to on, a global temporary table GLOBAL is created.
If enable_global_temp_table is set to off, a local temporary table LOCAL is created. You can also specify keyword LOCAL to reach the same effect.• If VOLATILE is specified, a temporary volatile table is created.
If default temptable_type is set to local, temporary tables created without keywords are local temporary tables. If default temptable_type is set to volatile, temporary tables created without keywords are volatile temporary tables. | <p>Example:</p> <pre>CREATE LOCAL TEMP TABLE temp_test (col1 INT, col2 DECIMAL(10,2));</pre> |

| Parameter | Description | Value Range or Example |
|------------------|--|---|
| TEMPORARY TEMP | <p>If TEMP or TEMPORARY is specified, the created table is a temporary table.</p> <p>Temporary tables are automatically dropped at the end of a session, or optionally at the end of the current transaction. Therefore, apart from CN and other CN errors connected by the current session, you can still create and use temporary table in the current session.</p> <p>Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are not advised to perform operations on temporary tables in DDL statements. TEMP is equivalent to TEMPORARY.</p> | For other restrictions of this parameter, see TEMPORARY TEMP Parameter Description . |
| IF NOT EXISTS | <p>If IF NOT EXISTS is specified, a table will be created if there is no table using the specified name. If there is already a table using the specified name, no error will be reported. A message will be displayed indicating that the table already exists, and the database will skip table creation.</p> | <p>Example:</p> <p>Use IF NOT EXISTS to prevent duplicate table creation.</p> <pre>CREATE TABLE IF NOT EXISTS test_t1 (col1 INT,col2 VARCHAR(30));</pre> |

| Parameter | Description | Value Range or Example |
|-------------|--|--|
| table_name | Specifies the name of the table to be created. | <ul style="list-style-type: none">• The table name must start with a letter or underscore (_) and can contain up to 63 characters. Only letters, digits, underscores (_), dollar signs (\$), and number signs (#) are allowed.• Chinese table names are supported. The number of supported characters depends on the database's character set encoding. For details, see "DWS Database Object Naming Rules" in the <i>Data Warehouse Service (DWS) Developer Guide</i>.• You are not advised to use a keyword as a table name. For details, see Keyword.• A table name enclosed in double quotation marks can contain spaces and special characters. However, you are not advised to use these characters in a table name because they may make it difficult to reference and use. In addition, they may be processed differently under different database compatibility modes. |
| column_name | Specifies the name of a column to be created in the new table. | The column name can contain a maximum of 63 characters, including letters, digits, underscores (_), dollar signs (\$), and number signs (#). It must start with a letter or underscore (_). |

| Parameter | Description | Value Range or Example |
|--|--|--|
| data_type | Specifies the data type of the column. | In a database compatible with Teradata or MySQL syntax, if the data type of a column is set to DATE, the DATE type is returned. Otherwise, the TIMESTAMP type is returned. |
| compress_mode | Specifies the compress option of the table, only available for row-store table. The option specifies the algorithm preferentially used by table columns. | DELTA, PREFIX, DICTIONARY, NUMSTR, or NOCOMPRESS |
| COLLATE collation | Assigns a collation to the column (which must be of a collatable data type). | If no collation is specified, the default collation is used. |
| LIKE source_table
[like_option ...] | Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints. | For details about LIKE constraints, see LIKE Parameter Description . |

| Parameter | Description | Value Range or Example |
|--|---|---|
| WITH | <p>Specifies an optional storage parameter for a table or an index.</p> <p>NOTE
Using Numeric of any precision to define column, specifies precision p and scale s. When precision and scale are not specified, the input will be displayed.</p> | <p>WITH contains many sub-parameters, which are classified as follows by scenario. You only need to pay attention to the parameters in specific scenarios.</p> <ul style="list-style-type: none">• For details about storage class parameters, see Table 21-142.• For details about parameters related to the partitioned table, see Table 21-143.• For details about the parameters related to the Delta table, see Table 21-144.• Parameters related to the Hstore table. For details, see Table 21-145.• For details about performance-related parameters, see Table 21-146. |
| ON COMMIT
{ PRESERVE
ROWS DELETE
ROWS } | <p>ON COMMIT determines what to do when you commit a temporary table creation operation. Global temporary tables support only the PRESERVE ROWS option.</p> | <ul style="list-style-type: none">• PRESERVE ROWS (Default): No special action is taken at the ends of transactions. The temporary table and its table data are unchanged.• DELETE ROWS: All rows in the temporary table will be deleted at the end of each transaction block. |

| Parameter | Description | Value Range or Example |
|-----------------------|--|---|
| COMPRESS NOCOMPRESS | If you specify COMPRESS in the CREATE TABLE statement, the compression feature is triggered in the case of a bulk INSERT operation. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If NOCOMPRESS is specified, the table is not compressed. | Default value:
NOCOMPRESS , tuple data is not compressed before storage. |
| DISTRIBUTE BY | <p>Specifies how the table is distributed or replicated between DNs.</p> <ul style="list-style-type: none">● REPLICATION: Each row in the table exists on all DNs, that is, each DN has complete table data.● ROUNDROBIN: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. This value is supported only in 8.1.2 or later.● HASH (column_name): Each row of the table will be placed into all the DNs based on the hash value of the specified column. <p>For details about other restrictions, see DISTRIBUTE BY Parameter Description.</p> | <p>Example:
Create a hash-distributed table.</p> <pre>CREATE TABLE test_t2(col1
INT,col2
VARCHAR(30))DISTRIBUTE BY
HASH(col1);</pre> |

| Parameter | Description | Value Range or Example |
|---|--|---|
| TO { GROUP
groupname
NODE
(nodename
[, ...]) } | TO GROUP specifies the Node Group in which the table is created. Currently, it cannot be used for HDFS tables. TO NODE is used for internal scale-out tools. | <p>In logical cluster mode, if TO GROUP is not specified, the table is created in the node group associated with the logical cluster user by default. If the user, such as the administrator or a common user, does not manage the logical cluster, by default the table is created in the first logical cluster, which is the logical cluster with the smallest OID in pgxc_group.</p> <p>If the node group specified by TO GROUP is a replication table node group, the table is created on all CNs and DN, but the replication table data is distributed only on the DN in the replication table node group.</p> <p>The Storage-compute decoupling 3.0 supports read-only logical clusters. If a user is not bound to any read-only logical clusters but sets TO GROUP to a logical cluster in a table creation statement, an error will be reported during table creation. If a user bound to a read-only logical cluster creates a table, the table will be created in the logical cluster specified by the GUC parameter default_storage_nodegroup. If default_storage_nodegroup is set to installation, tables will be created in the first logical cluster.</p> |

| Parameter | Description | Value Range or Example |
|-------------------------------|--|--|
| COMMENT [=]
'text' | The COMMENT clause can specify table comments during table creation. | Example:
Add a comment for a table.
<pre>CREATE TABLE table_user(user_id INT,username VARCHAR(50))
COMMENT 'user Information';</pre> |
| COMMENT 'text' | The COMMENT clause can specify a comment for a column. | Example:
Add a comment for a specified column.
<pre>CREATE TABLE table_users(user_id INT COMMENT 'employee id',username VARCHAR(50));</pre> |
| CONSTRAINT
constraint_name | <p>Specifies a name for a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an insert or update operation to succeed.</p> <p>There are two ways to define constraints:</p> <ul style="list-style-type: none">• A column constraint is defined as part of a column definition, and it is bound to a particular column.• A table constraint is not bound to any particular columns but can apply to more than one column. | Example:
Create the table_c_products table and define the primary key constraint. The primary key of the table consists of C_CUSTKEY and C_NAME .
<pre>CREATE TABLE table_c_products
(
 C_CUSTKEY BIGINT,
 C_NAME VARCHAR(25) ,
 C_ADDRESS VARCHAR(40),
 C_NATIONKEY INT,
 C_PHONE CHAR(15) ,
 C_ACCTBAL DECIMAL(15,2),
 CONSTRAINT C_CUSTKEY_KEY
 PRIMARY
 KEY(C_CUSTKEY,C_NAME)
)
DISTRIBUTE BY
HASH(C_CUSTKEY,C_NAME);</pre> |
| NOT NULL | Indicates that the column is not allowed to contain NULL values. | - |
| NULL | <p>The column is allowed to contain NULL values. This is the default setting.</p> <p>This clause is only provided for compatibility with non-standard SQL databases. You are advised not to use this clause.</p> | - |

| Parameter | Description | Value Range or Example |
|-------------------------|--|---|
| CHECK
(expression) | <p>Specifies an expression producing a Boolean result which new or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to TRUE or UNKNOWN succeed. If any row of an insert or update operation produces a FALSE result, an error exception is raised and the insert or update does not alter the database.</p> <p>A check constraint specified as a column constraint should reference only the column's values, while an expression appearing in a table constraint can reference multiple columns.</p> | <p><>NULL and !=NULL are invalid in an expression. Change them to IS NOT NULL.</p> |
| DEFAULT
default_expr | <p>Assigns a default data value for a column.</p> | <p>The value can be any variable-free expressions (Subqueries and cross-references to other columns in the current table are not allowed). The data type of the default expression must match the data type of the column.</p> <p>The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is NULL.</p> |

| Parameter | Description | Value Range or Example |
|-----------------------------|--|---|
| ON UPDATE
on_update_expr | <p>The ON UPDATE clause specifies a timestamp function for a column. Ensure that the data type of the column for which the ON UPDATE clause specifies a timestamp function is timestamp or timestamptz.</p> <p>When an SQL statement containing the UPDATE operation is executed, this column is automatically updated to the time specified by the timestamp function.</p> | <p>on_update_expr supports only CURRENT_TIMESTAMP, CURRENT_TIME, CURRENT_DATE, LOCALTIME, and LOCALTIMESTAMP.</p> |

| Parameter | Description | Value Range or Example |
|---|--|---|
| UNIQUE [NULLS
[NOT] DISTINCT
 NULLS IGNORE]
index_parameters

UNIQUE [NULLS
[NOT] DISTINCT
 NULLS IGNORE]
(column_name
[, ...])
index_parameters | <p>Specifies that a group of one or more columns of a table can contain only unique values.</p> <p>The [NULLS [NOT] DISTINCT NULLS IGNORE] field is used to specify how to process null values in the index column of the Unique index.</p> | <p>Default value: This parameter is left empty by default. NULL values can be inserted repeatedly.</p> <p>When the inserted data is compared with the original data in the table, the NULL value can be processed in any of the following ways:</p> <ul style="list-style-type: none">• NULLS DISTINCT: NULL values are unequal and can be inserted repeatedly.• NULLS NOT DISTINCT: NULL values are equal. If all index columns are NULL, NULL values cannot be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different.• NULLS IGNORE: NULL values are skipped during the equivalent comparison. If all index columns are NULL, NULL values can be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different. <p>The following table Table 21-148 lists the behaviors of the three processing modes.</p> <p>NOTE
If DISTRIBUTE BY REPLICATION is not specified, the column table that contains only unique values must contain distribution columns.</p> |

| Parameter | Description | Value Range or Example |
|---|--|--|
| PRIMARY KEY
index_parameters
PRIMARY KEY
(column_name
[, ...])
index_parameters | <p>Specifies the primary key constraint specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.</p> <p>Only one primary key can be specified for a table.</p> <p>CAUTION
If DISTRIBUTE BY REPLICATION is not specified, the column set with a primary key constraint must contain distributed columns.</p> | <p>Example:</p> <p>Create the products table, set the data type of product_no to integer and set it as the primary key.</p> <pre>CREATE TABLE products
(product_no integer PRIMARY
KEY,
name text,
price numeric
)
DISTRIBUTE BY
HASH(product_no);</pre> |
| REFERENCES
reftable
[(refcolumn)] | <p>The foreign key constraint requires that the group consisting of one or more columns in the new table should contain and match only the referenced column values in the referenced table. The referenced column should be the only column or primary key in the referenced table.</p> <p>DWS does not check foreign key constraints. When using foreign key constraints, you need to use the check_foreign_key_constraint function to check whether the data in the foreign key table meets the foreign key constraints.</p> | - |
| DEFERRABLE
NOT DEFERRABLE | <p>Controls whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the SET CONSTRAINTS command.</p> | <p>NOT DEFERRABLE is the default value. Currently, only UNIQUE and PRIMARY KEY constraints of row-store tables accept this clause. All the other constraints are not deferrable.</p> |

| Parameter | Description | Value Range or Example |
|--|---|---|
| PARTIAL CLUSTER KEY | Specifies a partial cluster key for storage. When importing data to a column-store table, you can perform local data sorting by specified columns (single or multiple). | Example:
Create the products_new table and define the partial cluster key on the product_no column.
<pre>CREATE TABLE products_new
(
 product_no integer,
 name text,
 price numeric,
 PARTIAL CLUSTER
 KEY(product_no)
) WITH (ORIENTATION =
COLUMN);</pre> |
| INITIALLY IMMEDIATE INITIALLY DEFERRED | If a constraint is deferrable, this clause specifies the default time to check the constraint. | <ul style="list-style-type: none">• If the constraint is INITIALLY IMMEDIATE (default value), it is checked after each statement.• If the constraint is INITIALLY DEFERRED, it is checked only at the end of the transaction.• The constraint check time can be altered using the SET CONSTRAINTS command. |

Table 21-142 WITH parameters (storage-related)

| Parameter | Description | Value Range |
|-------------|--|--|
| ORIENTATION | <p>Specifies the storage mode (row-store, column-store) for table data. This parameter cannot be modified once it is set.</p> <ul style="list-style-type: none">Row-store tables are suitable for OLTP workloads. There are many interactive transactions in such tables. An interaction involves multiple columns in a table.Column-store tables are suitable for data warehouse workloads. There are a large amount of aggregation calculation and few column operations on these tables. | <ul style="list-style-type: none">ROW indicates that table data is stored in rows.COLUMN indicates that the data is stored in columns. <p>Default value: ROW (row-store)</p> <p>NOTE
To create a column-store table, explicitly set orientation to column. When creating a local table in the storage-compute decoupling edition with all data stored on EVS disks, explicitly set colversion to 2.0.</p> |

| Parameter | Description | Value Range |
|---------------|--|--|
| COMPRESSION | Specifies the compression level of the table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time, and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution characteristics of loading table data. | <p>This parameter is available only for column-store tables. Value: LOW (default value), MIDDLE, or HIGH.</p> <p>For details about the compression algorithms provided by DWS, see Table 21-147.</p> <p>NOTE</p> <ul style="list-style-type: none">Currently, row-store table compression is not supported.When migrating data from ORC to DWS column-store tables, low-level compression increases the backup size by 1.5 to 2 times compared to ORC, whereas high-level compression results in a similar backup size. It is crucial to take the conversion relationship into account while setting up a DWS cluster.The middle compression of column-stores uses dictionary compression. For data not suitable for dictionary compression, the file size after middle compression may be greater than that of after low compression. |
| COMPRESSLEVEL | <p>Specifies the compression level of the table data. It determines the compression ratio and time.</p> <p>This divides a compression level into sublevels, providing you with more choices for compression rate and duration. As the value becomes greater, the compression rate becomes higher and duration longer at the same compression level. The parameter is only valid for column-store tables.</p> | <p>0~3</p> <p>Default value: 0</p> |

| Parameter | Description | Value Range |
|----------------------|--|---|
| MAX_BATCHROW | Specifies the maximum of a storage unit during data loading process. The parameter is only valid for column-store tables. | 10000 to 60000
Default value: 60,000 |
| PARTIAL_CLUSTER_ROWS | Specifies the number of records to be partial cluster stored during data loading process. The parameter is only valid for column-store tables. | 600000 to 2147483647
Default value: 4,200,000 |

| Parameter | Description | Value Range |
|------------|---|---|
| COLVERSION | Specifies the version of the column-store format. You can switch between different storage formats. | <ul style="list-style-type: none">• 1.0: Each column in a column-store table is stored in a separate file. The file name is relfilenode.C1.0, relfilenode.C2.0, relfilenode.C3.0, or similar.• 2.0: All columns of a column-store table are combined and stored in a file. The file is named relfilenode.C1.0.• 3.0: Each column of a column-store table is stored in a file. The file is stored in the OBS file system and named C1_fileid.0. <p>Default value: The default value for the storage-compute coupled version is 2.0, while for the storage-compute decoupling version, it is 3.0.</p> <p>The value of COLVERSION can only be set to 2.0 for OBS multi-temperature tables.</p> |

| Parameter | Description | Value Range |
|-----------|-------------|--|
| | | <p>NOTE</p> <ul style="list-style-type: none">For clusters of version 8.1.0, the default value of this parameter is 1.0. For clusters of version 8.1.1 or later, the default value of this parameter is 2.0. If the cluster version is upgraded from 8.1.0 to 8.1.1 or later, the default value of this parameter changes from 1.0 to 2.0.When creating a column-store table, set COLVERSION to 2.0. Compared with the 1.0 storage format, the performance is significantly improved:<ul style="list-style-type: none">The time required for creating a column-store wide table is significantly reduced.In the Roach data backup scenario, the backup time is significantly reduced.The build and catch up time is greatly reduced.The storage-compute decoupling version is compatible with all column-store versions. When creating a table, you need to explicitly set colversion to 1.0, 2.0, or 3.0. If colversion is set to 3.0, a table in decoupled storage and computing mode is created. If colversion is not explicitly specified, a column-store table of version 3.0 is created by default. When creating a table with decoupled storage and compute nodes, set colversion to 3.0 and set orientation to column.V3 storage-compute decoupling tables do not support colversion switching using ALTER TABLE, for example, from 2.0 to 3.0. |

| Parameter | Description | Value Range |
|--|---|---|
| cache_policy
(This parameter is supported only in 9.0.2 and later versions where the storage and compute nodes are decoupled.) | Specifies the cache mode of tables or partitioned tables (disks). If one of the following values is specified in the cache policy, hot cache is used. Otherwise, cold cache is used. Hot cache occupies more space than cold cache and uses more complex replacement policies. | <ul style="list-style-type: none">• ALL: Hot cache is used for the entire table.• NONE: Cold cache is used for the entire table.• HPN: The first <i>N</i> partitions in a partitioned table use hot cache. The rest of the partitions use cold cache.• HPL: <i>P1, P2,</i> In a partitioned table, the specified partitions use hot cache. The rest of the partitions use cold cache. <p>The default value is ALL.</p> <p>NOTE</p> <ul style="list-style-type: none">• For foreign tables and non-partitioned tables, only the ALL and NONE cache policies are supported.• Only range-partitioned and list-partitioned internal tables support HPN and HPL cache policies. |
| enable_disaster_cs_tore | Specifies whether fine-grained DR will be enabled for column-store tables. This parameter is available only for common column-store tables whose COLVERSION is 2.0. It is unavailable for HStore tables, row-store tables, temporary tables, unlogged tables, foreign tables, time series tables, cold and hot tables, materialized views, and delta tables, and cannot be enabled together with enable_hstore and enable_delta . This parameter is supported by version 8.2.0.100 or later clusters. | - |

| Parameter | Description | Value Range |
|---------------------------------------|---|---|
| <code>fine_disaster_table_role</code> | <p>This parameter has been discarded in version 8.2.1 and is reserved for compatibility with earlier versions. This parameter is invalid in the current version.</p> <p>Specifies whether the fine-grained DR table will be set as a primary or secondary table. This parameter can be true only when the enable_disaster_cstore parameter has been set to true.</p> | <ul style="list-style-type: none">• primary: Specifies the primary fine-grained DR table.• standby: Specifies the standby fine-grained DR table. |
| <code>on_commit_preserve_rows</code> | <p>It is similar to ON COMMIT { PRESERVE ROWS DELETE ROWS }. The two parameters cannot be specified at the same time. This parameter is used only for global temporary tables.</p> | The default value is true . |

Table 21-143 WITH parameters (related to partition tables)

| Parameter | Description | Value Range |
|---------------------|---|--|
| <code>TTL</code> | Schedules the partition deletion tasks in a partitioned table. By default, no partition deletion task is created. | 1 hour ~ 100 years |
| <code>PERIOD</code> | Schedules the partition creation tasks in a partitioned table. If TTL has been configured, PERIOD cannot be greater than TTL . | 1 hour ~ 100 years
Default value: 1 day |

| Parameter | Description | Value Range |
|-------------|--|---|
| time_format | You can use auto-increment and decrement partitions for INT4 , INT8 , VARCHAR , and TEXT columns, which are commonly used for storing time-related data. The time_format option is applicable only when the partition key is INT4 , INT8 , VARCHAR , or TEXT and a period is specified. This is supported only by clusters of version 9.1.0.200 or later. | <ol style="list-style-type: none">1. The value of time_format must comply with the PostgreSQL specifications, for example, yyyymmdd.2. Restrictions on time_format differ by partition key type.
VARCHAR/TEXT:<ul style="list-style-type: none">• The precision can be accurate to seconds.• The value cannot contain letters, for example, month, am, and pm.• The time must be arranged in descending order, for example, year, month, day, hour, minute, and second.INT4/INT8:<ul style="list-style-type: none">• The precision can be accurate to hours.• The value can contain only Y, M, D, and HH24.• The time must be arranged in descending order, for example, year, month, day, and hour.3. ALTER restrictions:<ul style="list-style-type: none">• The set operation is not supported.• When period is reset (indicating that automatic partitioning is disabled and a message is displayed), you can reset this option. |

Table 21-144 WITH parameters (related to the Delta table)

| Parameter | Description | Value Range |
|------------------------|--|---|
| enable_delta | Specifies whether to enable delta tables in column-store tables. The parameter is only valid for column-store tables. If COLVERSION is set to 3.0 , enable_delta cannot be turned on because this parameter is not supported by V3 tables.

Using column-store tables with delta tables is not recommended. This may cause disk bloat and performance deterioration due to delayed merge. | Default value: off |
| DELTAROW_THRE
SHOLD | Specifies the upper limit of to-be-imported rows for triggering the data import to a delta table when data is to be imported to a column-store table. This parameter takes effect only if the enable_delta table parameter is set to on . The parameter is only valid for column-store tables. | 0 to 60000
Default value: 6000 |

Table 21-145 WITH parameters (related to Hstore tables)

| Parameter | Description | Value Range |
|---------------|--|--|
| enable_hstore | Specifies whether an HStore table will be created (based on column-store tables). The parameter is only valid for column-store tables. This parameter is supported by version 8.2.0.100 or later clusters. If COLVERSION is set to 3.0 , enable_delta cannot be turned on because this parameter is not supported by V3 tables. | Default value: off
NOTE
If this parameter is enabled, the following GUC parameters must be set to ensure that H-Store tables are cleared.

Set autovacuum to on , autovacuum_max_workers to 6 , and autovacuum_max_workers_hstore to 3 . |

| Parameter | Description | Value Range |
|-----------------------|--|--|
| enable_hstore_opt | If the enable_hstore_opt table-level parameter is enabled, the enable_hstore table-level parameter is also enabled by default. This parameter is supported only in cluster 8.3.0 and later versions. | Default value: false |
| secondary_part_column | Specifies the name of a level-2 partition column in a column-store table. Only one column can be specified as the level-2 partition column. This parameter applies only to HStore column-store tables. This parameter is supported only by clusters of version 8.3.0 or later. | <ul style="list-style-type: none">• The column specified as a level-2 partition column cannot be deleted or modified.• The level-2 partition column can be specified only when a table is created. After a table is created, the level-2 partition column cannot be modified.• You are not advised to specify a distribution column as a level-2 partition column.• The level-2 partition column determines how the table is logically split into hash partitions on DNs, which enhances the query performance for that column. |
| secondary_part_num | Specifies the number of level-2 partitions in a column-store table. This parameter applies only to HStore column-store tables. This parameter is supported only by clusters of version 8.3.0 or later. | 1~32
Default value: 8
NOTICE <ul style="list-style-type: none">• This parameter can be specified only when secondary_part_column is specified.• The number of level-2 partitions can be specified only when a table is created and cannot be modified after the table is created.• You are not advised to change the default value, which may affect the import and query performance. |

| Parameter | Description | Value Range |
|----------------|---|-------------|
| bitmap_columns | The bitmap index is only applicable to the new HStore (hstore_opt) tables. To generate the bitmap index mapping, you need to enable the enable_hstore_opt table-level parameter and set bitmap_columns to Specified Columns . This parameter is supported only by clusters of version 8.3.0 or later. | - |

Table 21-146 WITH parameters (other performance-related parameters)

| Parameter | Description | Value Range |
|------------|---|-------------|
| FILLFACTOR | The fillfactor of a table is a percentage between 10 and 100. 100 (complete packing) is the default value. When a smaller fillfactor is specified, INSERT operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose records are never updated, setting the fillfactor to 100 (complete packing) is the appropriate choice, but in heavily updated tables smaller fillfactors are appropriate. The parameter has no meaning for column-store tables. | 10~100 |

| Parameter | Description | Value Range |
|---------------------|---|--|
| analyze_mode | Specifies the mode of table-level auto-analyze. | <ul style="list-style-type: none">• frozen: disables all ANALYZE operations (dynamic sampling can still be triggered when no statistics are collected).• backend: allows only ANALYZE triggered by AUTOVACUUM polling.• runtime: allows only runtime ANALYZE triggered by the optimizer.• all: Both backend and runtime AUTO-ANALYZE can be triggered. Default value: all |
| incremental_analyze | Specifies whether to enable the incremental analyze mode for partitioned tables. This parameter is valid only for partitioned tables and cannot be set for replicated tables. This is supported only by clusters of version 9.1.0.100 or later. | Default value: false |

| Parameter | Description | Value Range |
|--------------------|--|--|
| enable_turbo_store | <p>Determines whether to create a turbo table (column-store tables). The parameter is only valid for column-store tables. It is available only for clusters of versions 9.1.0.100 and later.</p> <p>NOTE</p> <ul style="list-style-type: none">Common column-store tables in version 3.0 do not support the turbo storage format, while HStore Opt tables in version 3.0 only support the turbo storage format.In version 2.0, there is no restriction on column-store tables.Currently, turbo tables do not support time series tables, delta tables, or lightweight update. | <p>Default value: off</p> <p>NOTE</p> <p>Turbo tables enhance the storage efficiency of numeric and varchar data types using the integers, leading to accelerated processing speeds for these types.</p> <ul style="list-style-type: none">For enhanced performance, it is advised to define the numeric type with precision and scale (p, s). The storage allocation is as follows:
For $p \leq 9$, 4-byte integers are used for storage. For $p \leq 18$, 8-byte integers are used. For $p \leq 37$, 16-byte integers are used. If $p > 37$ or remain unspecified, a variable-length format is used, which is less space-efficient and yields suboptimal performance.If the maximum length (n) of the varchar type is less than or equal to 2 bytes, 2-byte integers are used for storage. If the maximum length (n) of the varchar type is less than or equal to 4 bytes, 4-byte integers are used for storage. If the maximum length (n) of the varchar type is less than or equal to 16 bytes, 16-byte integers are used for storage. Limit VARCHAR columns in GROUP BY or HASH JOIN operations to 16 bytes or less to enable faster integer-based storage and improve performance. |

| Parameter | Description | Value Range |
|----------------------------------|--|---|
| SKIP_FPI_HINT | Indicates whether to skip the hint bits operation when the full-page writes (FPW) log needs to be written during sequential scanning. | Default value: false
NOTE
If SKIP_FPI_HINT is set to true and the checkpoint operation is performed on a table, no Xlog will be generated when the table is sequentially scanned. This applies to intermediate tables that are queried less frequently, reducing the size of Xlogs and improving query performance. |
| enable_column_autovacuum_garbage | Determines whether to enable CU rewriting logic for column-store tables using AUTOVACUUM. This parameter is supported only by clusters of version 8.2.1.100 or later.

There is a low probability that an error is reported when lightweight UPDATE and AUTOVACUUM are executed concurrently. You can set the table-level parameter to off to avoid this problem. | Default value: true |

Table 21-147 Compression algorithms for column-based storage

| COMPRESSION | NUMERIC | STRING | INT |
|-------------|--|--------------------------------------|---|
| LOW | Delta compression + RLE compression | LZ4 compression | Delta compression (RLE is optional.) |
| MIDDLE | Delta compression + RLE compression + LZ4 compression | dict compression or LZ4 compression | Delta compression or LZ4 compression (RLE is optional) |
| HIGH | Delta compression + RLE compression + zlib compression | dict compression or zlib compression | Delta compression or zlib compression (RLE is optional) |

Table 21-148 Processing of NULL values in index columns in unique indexes

| Constraint | All Index Columns Are NULL | Some Index Columns Are NULL. |
|--------------------|--------------------------------|--|
| NULLS DISTINCT | Can be inserted repeatedly. | Can be inserted repeatedly. |
| NULLS NOT DISTINCT | Cannot be inserted repeatedly. | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |
| NULLS IGNORE | Can be inserted repeatedly. | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |

TEMPORARY | TEMP Parameter Description

- Local or volatile temporary tables are visible to the current session through schema of the **pg_temp** start. Users should not delete schema started with **pg_temp**, **pg_toast_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table and the schema of the specified table starts with **pg_temp_**, the table is created as a temporary table.
- Similar to common tables, all metadata of local temporary tables is stored in system catalogs. Volatile temporary tables store table structure metadata in memory, except the schema metadata. Compared with local temporary tables, volatile temporary tables have the following constraints:
 - After a CN or DN is restarted, data in its memory will be lost, and accordingly, volatile temporary tables on it will become invalid.
 - Currently, volatile temporary tables do not support table structure modification, such as **ALTER** and **GRANT**.
 - Volatile temporary tables and local temporary tables share temporary schemas. Therefore, in the same session, the **VOLATILE** temporary table and local temporary table cannot have the same name.
 - Volatile temporary table information is not stored in system catalogs. Therefore, Volatile metadata cannot be queried by running DML statements in system catalogs.
 - Volatile temporary tables support only common row-store and column-store tables. Delta tables, time series tables, and cold and hot tables are not supported.
 - Views cannot be created based on volatile temporary tables.
 - A tablespace cannot be specified when a temporary table is created. The default tablespace of a volatile temporary table is **pg_volatile**.

- The following constraints cannot be specified when a volatile temporary table is created: CHECK, UNIQUE, PRIMARY KEY, TRIGGER, EXCLUDE, and PARTIAL CLUSTER.
- Similar to common tables, all metadata in global temporary tables is stored in system catalogs.
 - The difference between a global temporary table and a local temporary table is that in a global temporary table, the metadata is not deleted when a session exits, but the session data is deleted. Data of different sessions is independent but shares the metadata of the same global temporary table.
 - A global temporary table has a schema that is similar to a regular table, unlike a local or volatile temporary table which has a schema starting with **pg_temp**. This means that a global temporary table can have the same name as a local or volatile temporary table.
 - Global temporary tables support only common row-store and column-store tables. Delta tables, time series tables, and cold and hot tables are not supported.
 - Operations cannot be performed on global temporary tables of other logical clusters.

LIKE Parameter Description

The new table and the source table are decoupled after creation is complete. Changes to the source table will not be applied to the new table, and it is not possible to include data of the new table in scans of the source table.

Columns and constraints copied by **LIKE** are not merged with the same name. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

- The default expressions or the **ON UPDATE** expressions are copied from the source table to the new table only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
- The **CHECK** constraints are copied from the source table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. **NOT NULL** constraints are always copied to the new table. These rules also apply to column constraints and table constraints.
- Any indexes on the source table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- **STORAGE** settings for the copied column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.
- If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the **PARTITION BY** clause. The default behavior is to exclude partition definition of the source table.

- If **INCLUDING REOPTIONS** is specified, the storage parameter (**WITH** clause of the source table) of the source table is copied to the new table. The default behavior is to exclude partition definition of the storage parameter of the source table.

 NOTE

PERIOD and **TTL** in the **WITH** clause are partition-related parameters. **LIKE INCLUDING REOPTIONS** will not be copied to the new table. To copy **LIKE INCLUDING REOPTIONS**, use **INCLUDING PARTITION**.

- If **INCLUDING DISTRIBUTION** is specified, the distribution information of the source table is copied to the new table, including distribution type and column, and the new table no longer use the **DISTRIBUTE BY** clause. The default behavior is to exclude distribution information of the source table.
- If **INCLUDING DROPCOLUMNS** is specified, the deleted column information in the source table is copied to the new table. By default, the deleted column information of the source table is not copied.
- **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, **INCLUDING REOPTIONS**, **INCLUDING DISTRIBUTION**, and **INCLUDING DROPCOLUMNS**.
- If **EXCLUDING** is specified, the specified parameters are not included.
- For an OBS multi-temperature table, all partitions of the new table are local hot partitions after **INCLUDING PARTITION** is specified.
- If **ALTER TABLE...ADD PARTITION** and **CREATE TABLE** are both executed for the source table, the number of partitions on different nodes may be inconsistent occasionally. As a result, an error is reported during the execution of **CREATE TABLE**. You are advised to execute **CREATE TABLE** again.

NOTICE

- If the source table contains a sequence with the **SERIAL**, **BIGSERIAL**, or **SMALLSERIAL** data type, or a column in the source table is a sequence by default and the sequence is created for this table by using **CREATE SEQUENCE... OWNED BY**, these sequences will not be copied to the new table, and another sequence specific to the new table will be created. This is different from earlier versions. To share a sequence between the source table and new table, create a shared sequence (do not use **OWNED BY**) and set a column in the source table to this sequence.
 - You are not advised to set a column in the source table to the sequence specific to another table especially when the table is distributed in specific node groups, because doing so may result in **CREATE TABLE ... LIKE** execution failures. In addition, doing so may cause the sequence to become invalid in the source sequence because the sequence will also be deleted from the source table when it is deleted from the table that the sequence is specific to. To share a sequence among multiple tables, you are advised to create a shared sequence for them.
-

DISTRIBUTE BY Parameter Description

Specifies how the table is distributed or replicated between DN.

Valid value:

- **REPLICATION**: Each row in the table exists on all DN, that is, each DN has complete table data.
- **ROUNDROBIN**: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. This value is supported only in 8.1.2 or later.
- **HASH (column_name)**: Each row of the table will be placed into all the DN based on the hash value of the specified column.

NOTE

- When **DISTRIBUTE BY HASH (column_name)** is specified, the primary key and its unique index must contain the **column_name** column.
- When **DISTRIBUTE BY HASH (column_name)** in a referenced table is specified, the foreign key of the reference table must contain the **column_name** column.
- If **TO GROUP** is set to a replication table node group (supported in 8.1.2 or later), **DISTRIBUTE BY** must be set to **REPLICATION**. If **DISTRIBUTE BY** is not specified, the created table is automatically set as a replication table.

Default value: determined by the GUC parameter **default_distribution_mode**

- When **default_distribution_mode** is set to **roundrobin**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 - a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - b. If the primary key or unique constraint is not included during table creation, round-robin distribution is selected.
- When **default_distribution_mode** is set to **hash**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 - a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - b. 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 columns, hash distribution is selected. The distribution column is the first column whose data type can be used as a distribution column.
 - c. 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 column exists, round-robin distribution is selected.

The following data types can be used as distribution columns:

- Integer types: **TINYINT**, **SMALLINT**, **INT**, **BIGINT**, and **NUMERIC/DECIMAL**
- Character types: **CHAR**, **BPCHAR**, **VARCHAR**, **VARCHAR2**, **NVARCHAR2**, and **TEXT**
- Date/time types: **DATE**, **TIME**, **TIMETZ**, **TIMESTAMP**, **TIMESTAMPTZ**, **INTERVAL**, and **SMALLDATETIME**

 NOTE

When you create a table, the choices of distribution keys and partition keys have major impact on SQL query performance. Therefore, choosing proper distribution column and partition key with strategies.

- Selecting an Appropriate Distribution Column

In the data distributed table using Hash, an appropriate distributed array should be used to distribute and store data on multiple DNs evenly, preventing data skew (uneven data distribution across several DNs). Determine the proper distribution column based on the following principles:

1. Determine whether data is skewed.

Connect to the database and run the following statements to check the number of tuples on each DN: Replace *tablename* with the actual name of the table to be analyzed.

```
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 based on the following principles:

2. Run the ALTER TABLE statement to adjust the distribution column. The rules for selecting a distribution column are as follows:

The column value of the distribution column should be discrete so that data can be evenly distributed on each DN. For example, you are advised to select the primary key of a table as the distribution column, and the ID card number as the distribution column in a personnel information table.

With the above principles met, you can select join conditions as distribution keys so that join tasks can be pushed down to DNs, reducing the amount of data transferred between the DNs.

3. If a proper distribution column cannot be found to make data evenly distributed on each DN, you can use the **REPLICATION** or **ROUNDROBIN** data distribution mode. The **REPLICATION** data distribution mode stores complete data on each DN. Therefore, if a table is large and no proper distribution column can be found, the **ROUNDROBIN** data distribution mode is recommended. The **ROUNDROBIN** data distribution mode is supported in 8.1.2 or later.

- Selecting appropriate partition keys

In range partitioning, the table is partitioned into ranges defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.

Modify partition keys to make the query result stored in the same or least partitions (partition pruning). Obtaining consecutive I/O to improve the query performance.

In actual services, time is used to filter query objects. Therefore, you can use time as a partition key, and change the key value based on the total data volume and single data query volume.

Example: Creating a Storage-Compute Coupled Table

Create the **test_t4** table.

```
CREATE TABLE IF NOT EXISTS test_t4 ( ---Create a table when there are no test_t4 tables in the database.
col_id INT PRIMARY KEY, ---Define an INTEGER column and set it as the primary key.
col_number INT NOT NULL, ---Define an INTEGER column and set it to not null.
col_date DATE NOT NULL, --- Define a DATE column and set it to non-null.
col_price NUMERIC(10,2), ---Define a NUMERIC column. The maximum number of integer digits is 10,
and the maximum number of decimal places is 2.
col_status TEXT ---Define a TEXT column.
)
WITH (ORIENTATION = COLUMN,COLVERSION=2.0,COMPRESSION=middle) --- Set the storage method of
a table to column storage and the compression level to MIDDLE.
```

```
DISTRIBUTE BY HASH (col_id)      ---Use hash distribution.  
COMMENT 'order information for test_table4'; ---Comment a table.
```

Example: Creating a Storage-Compute Decoupled V3 Table

Create a V3 table (**test_t5**) with decoupled storage and compute (supported only in the storage-compute decoupling 3.0 version).

```
CREATE TABLE test_t5  
(  
  id integer not null,  
  data integer,  
  age integer  
)  
WITH (ORIENTATION =COLUMN, COLVERSION =3.0)  
DISTRIBUTE BY ROUNDROBIN;
```

Specify the cache policy when creating a table (supported only in the storage-compute decoupling 3.0 version).

```
CREATE TABLE sports  
(  
  N_NATIONKEY INT NOT NULL  
  , N_NAME CHAR(25) NOT NULL  
  , N_REGIONKEY INT NOT NULL  
  , N_COMMENT VARCHAR(152)  
) WITH (orientation = column, colversion = 3.0, cache_policy = 'HPL: Balls, Basketball')  
tablespace cu_obs_tbs  
DISTRIBUTE BY ROUNDROBIN  
partition by list(N_NAME)  
(  
  partition Balls values ('Basketball', 'football', 'badminton'),  
  partition Athletics values ('High jump', 'long jump', 'javelin'),  
  partition Water_Sports values ('Surfing', 'diving', 'swimming'),  
  partition Shooting values ('air guns', 'Rifles', 'archery'),  
  partition rest values (DEFAULT)  
);
```

Example: Defining Constraints When Creating a Table

Define **PRIMARY KEY** constraints for one or more columns of a table.

```
CREATE TABLE CUSTOMER  
(  
  C_CUSTKEY BIGINT ,  
  C_NAME VARCHAR(25) ,  
  C_ADDRESS VARCHAR(40) ,  
  C_NATIONKEY INT ,  
  C_PHONE CHAR(15) ,  
  C_ACCTBAL DECIMAL(15,2) ,  
  CONSTRAINT C_CUSTKEY_KEY PRIMARY KEY(C_CUSTKEY,C_NAME)  
)  
DISTRIBUTE BY HASH(C_CUSTKEY,C_NAME);
```

Define a **UNIQUE** constraint for a table.

```
CREATE TABLE CUSTOMER  
(  
  C_CUSTKEY BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY,  
  C_NAME VARCHAR(25) ,  
  C_ADDRESS VARCHAR(40) ,  
  C_NATIONKEY INT ,  
  C_PHONE CHAR(15) ,  
  C_ACCTBAL DECIMAL(15,2)  
)  
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Define a **CHECK** constraint for a table.

```
CREATE TABLE CUSTOMER
(
  C_CUSTKEY BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY,
  C_NAME VARCHAR(25) ,
  C_ADDRESS VARCHAR(40) ,
  C_NATIONKEY INT ,
  CONSTRAINT C_CUSTKEY_KEY2 CHECK(C_CUSTKEY > 0 AND C_NAME <> '')
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Define a **CHECK** constraint for a column.

```
CREATE TABLE CUSTOMER
(
  C_CUSTKEY BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY ,
  C_NAME VARCHAR(25) ,
  C_ADDRESS VARCHAR(40) ,
  C_NATIONKEY INT NOT NULL CHECK (C_NATIONKEY > 0)
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Use **DEFAULT** to declare the default value **GA** for the column **col_STATE**.

```
CREATE TABLE test_t5
(
  col_SK INTEGER NOT NULL,
  col_ID CHAR(16) NOT NULL,
  colE_NAME VARCHAR(20) UNIQUE DEFERRABLE,
  col_WAREHOUSE INTEGER,
  col_COUNTY VARCHAR(30),
  col_STATE CHAR(2) DEFAULT 'GA',
  col_ZIP CHAR(10)
);
```

Example: Creating a Column-Store Table and Specifying the Compression Level

Create the column-store table **customer_address** and set the data compression level to **HIGH**.

```
CREATE TABLE 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)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH,COLVERSION=2.0)
DISTRIBUTE BY HASH (ca_address_sk);
```

Example: Using CREATE TABLE LIKE to Create a Table

Use CREATE TABLE LIKE to quickly copy the schema of the existing table **test_t6** and create the table **test_t7**.

```
CREATE TABLE IF NOT EXISTS test_t6
(
  col_id INT PRIMARY KEY,
  col_number INT NOT NULL,
  col_date DATE NOT NULL,
  col_price NUMERIC(10,2),
  col_status TEXT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=middle)
```

```
DISTRIBUTE BY HASH (col_id);  
CREATE TABLE test_t7 (LIKE test_t6 INCLUDING ALL);
```

Helpful Links

[ALTER TABLE](#), [12.101-RENAME TABLE](#), and [DROP TABLE](#)

21.13.4.2 CREATE TABLE PARTITION

Function

CREATE TABLE PARTITION creates a partitioned table. A partitioned table is a logical table that is divided into segments, called partitions whose data is stored on different physical blocks.

Common partitioning strategies define data ranges based on one or more columns. Each partition stores data within a range. These columns are called partition keys.

Currently, DWS row-store and column-store tables support range partitioning and list partitioning.

- Range partitioning
A table is partitioned into ranges defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.
 - Core feature: Data is distributed by consecutive intervals. It is used in scenarios where data is ordered and retrieved by range.
 - Scenario: Partition keys increase linearly or have continuous intervals. Data query is usually performed around a certain range, and new data naturally falls into a new range. For example, data about orders is partitioned by time range, and data about user levels is partitioned by value range.
 - Range partitioning maps data to partitions based on ranges of values of the partitioning key that you establish for each partition. This is the most commonly used partitioning policy. Currently, range partitioning only allows the use of the range partitioning policy.
- List partitioning
List partitioning allocates records to partitions based on the key values in each partition. The key values do not overlap in different partitions. Each set of key values gets its own partition for storing related data. List partitioning is supported only by clusters of version 8.1.3 or later.
 - Core feature: Data is divided based on discrete enumerated values, which are used as fixed classification category or high-frequency filter criteria.
 - Scenario: Partition key values are fixed and discrete. Data query is usually based on a certain category, and key values do not increase irregularly. For example, the user information table is partitioned by province, and each province corresponds to a partition.
 - In list partitioning, data is mapped to a created partition based on the partition key value. If the data can be mapped to, it is inserted into the specific partition. If it cannot be mapped to, error messages are returned.

Major Differences Between Range Partitioning and List Partitioning

Table 21-149 Range partitioning and list partitioning

| Dimension | Range Partitioning | List Partitioning |
|--------------------|---|---|
| Data distribution | Continuous range (time and value) | Discrete enumerations values (classification and status) |
| Query optimization | Range filter (BETWEEN, >, <) | Exact match (= and IN) |
| Scenario | Ranges need to be added or deleted regularly (for example, rolling by month). | Fixed categories need to be maintained (such as regions and types). |
| Typical Case | Log, time series, and numeric archiving | Regional division, status isolation, and multi-tenant data |

Advantages of Partitioning

- The performance of some types of queries can be greatly improved, especially when rows with high access rates in a table are located in a single partition or a few partitions. Partitioning narrows the range of data search and improves data access efficiency.
- When queries or updates access a large percentage of a single partition, performance can be improved by taking advantage of sequential scan of that partition instead of reads scattered across the whole table.
- Bulk loads and deletion can be performed by adding or removing partitions, if that requirement is planned into the partitioning design. It also entirely avoids the **VACUUM** overhead caused by a bulk **DELETE** (only for range partitioning).
- Generally, if there are more than 10 million rows of data on a single DN, you are advised to create a partitioned table and ensure that the data volume on a single DN ranges from 1 million to 10 million.

Precautions

- A single table can have up to 32,767 partitions with a total of 32,767 boundary values. For better performance, keep this value at or below 1,000.
- A partitioned table supports unique and primary key constraints. The constraint keys of these constraints contain all partition keys.

Syntax

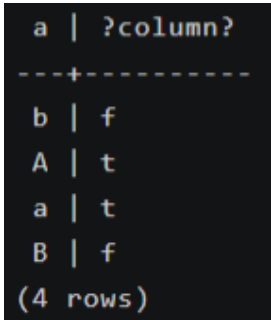
```
CREATE TABLE [ IF NOT EXISTS ] partition_table_name  
( [  
    { column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]  
    | table_constraint
```

```
| LIKE source_table [ like_option [...] ] [, ... ]  
] )  
[ WITH ( {storage_parameter = value} [, ...] ) ]  
[ COMPRESS | NOCOMPRESS ]  
[ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { [ HASH ] ( column_name ) } } ]  
[ TO { GROUP groupname | NODE ( nodename [, ...] ) } ]  
PARTITION BY {  
    {VALUES (partition_key)} |  
    {RANGE (partition_key) ( partition_less_than_item [, ...] ) } |  
    {RANGE (partition_key) ( partition_start_end_item [, ...] ) } |  
    {LIST (partition_key) (list_partition_item [, ...])}  
} [ { ENABLE | DISABLE } ROW MOVEMENT ];
```

CREATE TABLE PARTITION Main Clause Parameters

Table 21-150 CREATE TABLE PARTITION main clause parameters

| Parameter | Description | Value Range or Example |
|----------------------|--|--|
| IF NOT EXISTS | Does not throw an error if a table with the same name exists. A notice is issued in this case. | - |
| partition_table_name | Name of the partitioned table | A string, which must comply with the naming convention. For details, see Identifier Naming Conventions . |
| column_name | Specifies the name of a column to be created in the new table. | A string, which must comply with the naming convention. For details, see Identifier Naming Conventions . |
| data_type | Specifies the data type of the column. | - |

| Parameter | Description | Value Range or Example |
|-------------------|---|--|
| COLLATE collation | <p>The COLLATE clause defines the collation rule of a column. The supported data types include char, varchar, text, nchar, and nvarchar. The collation rule significantly impacts the sequence of query results, particularly when using SELECT ... ORDER BY. If no collation rule is specified, the default collation rule C (case-sensitive) is used.</p> <p>Common collation rules:</p> <ul style="list-style-type: none">• C: The collation uses the byte order of the binary code and is sensitive to case, accents, and special characters. Data can indeed be compared based on its ASCII code values.• case_insensitive: The collation is case-insensitive. It is supported by clusters of version 8.1.3.x or later.• en_US.utf8: The collation uses the American English with a UTF-8 encoding and is case insensitive. <p>You can query collation rules supported by DWS from the PG_COLLATION system catalog.</p> <pre>SELECT * FROM pg_collation;</pre> <p>To check the default collation rule of the current database, run the following statement:</p> <pre>SELECT pg_database.datcollate AS current_collation FROM pg_catalog.pg_database WHERE pg_database.datname = pg_catalog.current_database();</pre> | <p>A case-insensitive collation rule for the values of a column</p> <pre>CREATE TABLE t1 (a text collate case_insensitive); INSERT INTO t1 values('a'),('A'), ('b'),('B'); SELECT a, a='a' FROM t1;</pre> <p>The following figure shows that A and a are equivalent.</p>  |

| Parameter | Description | Value Range or Example |
|--|--|---|
| CONSTRAINT
constraint_name | <p>Specifies a name for a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an insert or update operation to succeed.</p> <p>There are two ways to define constraints:</p> <ul style="list-style-type: none">• A column constraint is defined as part of a column definition, and it is bound to a particular column.• A table constraint is not bound to any particular columns but can apply to more than one column. | For details about the supported constraints, see Table 21-151 . |
| LIKE source_table
[like_option ...] | Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints. | For details about the supported constraints, see LIKE Parameters . |
| WITH | Specifies an optional storage parameter for a table or an index. | For details about the supported WITH parameters, see Table 21-152 . |
| COMPRESS /
NOCOMPRESS | Specifies the keyword COMPRESS during the creation of a table, so that the compression feature is triggered in the case of a bulk INSERT operation. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If NOCOMPRESS is specified, the table is not compressed. | Default value: NOCOMPRESS , tuple data is not compressed before storage. |

| Parameter | Description | Value Range or Example |
|---|--|---|
| DISTRIBUTE BY | Specifies how the table is distributed or replicated between DNs. | <ul style="list-style-type: none">• REPLICATION: Each row in the table exists on all DNs, that is, each DN has complete table data.• ROUNDROBIN: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. This value is supported only in 8.1.2 or later.• HASH (column_name): Each row of the table will be placed into all the DNs based on the hash value of the specified column. <p>For details about other restrictions, see DISTRIBUTE BY Parameter Description.</p> <p>CAUTION</p> <ul style="list-style-type: none">• When DISTRIBUTE BY HASH (column_name) is specified, the primary key and its unique index must contain the column_name column.• When DISTRIBUTE BY HASH (column_name) in a referenced table is specified, the foreign key of the reference table must contain the column_name column. |
| TO { GROUP groupname NODE (nodename [, ...]) } | TO GROUP specifies the Node Group in which the table is created. Currently, it cannot be used for HDFS tables. TO NODE is used for internal scale-out tools. | - |

| Parameter | Description | Value Range or Example |
|-----------------------------------|--|--|
| { ENABLE DISABLE } ROW MOVEMENT | <p>Specifies whether to relocate a row within a table. If the new value of a partition key in a row belongs to another partition, an error message is displayed or the row is moved to the corresponding partition.</p> <ul style="list-style-type: none">• ENABLE: The row is moved to the corresponding partition. If partition keys are frequently updated, enable this parameter.• DISABLE: The row is not moved to the corresponding partition. If partition keys are updated, an error is reported. If data cross-partition update is strictly prohibited, disable this parameter. <p>The default value is DISABLE.</p> <p>Suggestions:</p> <ul style="list-style-type: none">• Scenario: To enable row movement, you need to change partition key values through UPDATE operations and can accept its impact on performance.• Performance: Enabling row movement will increase the overhead of UPDATE operations. Exercise caution when enabling this parameter in high-concurrency scenarios.• Alternative solution: If you need to frequently adjust the partition ownership, delete the original row and then insert a new row, or exchange partitions. | <p>The date column (partition key) of a partitioned table is partitioned by quarter into p_2023q1 and p_2023q2. The value 2023-02-15 of a row belongs to the partition p_2023q1 in the first quarter. After the value is updated to 2023-05-15, you can configure ROW MOVEMENT to determine whether to move the data in this row.</p> <ul style="list-style-type: none">• ENABLE: The row movement is enabled. The data is moved to the new partition p_2023q2.• DISABLE: The row movement is disabled. The system reports an error. <p>For details about examples, see Examples: Enabling and Disabling ROW MOVEMENT.</p> |

| Parameter | Description | Value Range or Example |
|-----------|---|------------------------|
| | CAUTION
To enable cross-partition update, you need to enable row movement. However, if SELECT FOR UPDATE is executed concurrently to query the partitioned table, the query results may be inconsistent. Therefore, exercise caution when performing this operation. | |

CONSTRAINT constraint_name Parameter

```
{ NOT NULL |  
  NULL |  
  CHECK ( expression ) |  
  DEFAULT default_expr |  
  UNIQUE [ NULLS [NOT] DISTINCT | NULLS IGNORE ] index_parameters |  
  PRIMARY KEY index_parameters }  
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

Table 21-151 CONSTRAINT constraint_name parameters

| Parameter | Description | Value Range or Example |
|-----------|--|------------------------|
| NOT NULL | Specifies that the column is not allowed to contain NULL values. | - |
| NULL | Specifies that the column can store NULL values. This is the default. This clause is only provided for compatibility with non-standard SQL databases. You are advised not to use this clause. | - |

| Parameter | Description | Value Range or Example |
|--|--|---|
| CHECK
(condition) [NO
INHERIT] | <p>Specifies an expression producing a Boolean result which new or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to TRUE or UNKNOWN succeed, or an error is reported and the execution fails.</p> <p>A check constraint specified as a column constraint should reference only the column's values, while an expression appearing in a table constraint can reference multiple columns.</p> <p>A constraint marked with NO INHERIT will not propagate to child tables.</p> | <p>Single-column constraints:</p> <p>The price must be a positive number.</p> <pre>CREATE TABLE products (
 product_id INT,
 price DECIMAL(10,2) CHECK
 (price > 0),
 discount DECIMAL(3,2));</pre> <p>Insert a data record where the price is a negative number.</p> <pre>INSERT INTO products
(product_id, price, discount)
VALUES (2, -10.00, 1.1);</pre> <p>The constraint is violated, so an error is reported and the insertion fails.</p> <pre>ERROR: dn_6003_6004: new row
for relation "products" violates
check constraint
"products_price_check"</pre> <p>Table constraints (on multiple columns):</p> <p>Create an inventory table with two columns available and reserved. One column indicates available quantity and the other column indicates reserved quantity. The available quantity must be greater than or equal to the reserved quantity.</p> <pre>CREATE TABLE inventory (
 item_id INT PRIMARY KEY,
 quantity_on_hand INT,
 quantity_reserved INT,
 CHECK (quantity_on_hand >=
 quantity_reserved));</pre> |

| Parameter | Description | Value Range or Example |
|-------------------------|--|------------------------|
| DEFAULT
default_expr | <p>Assigns a default data value for a column. The value can be any variable-free expressions (Subqueries and cross-references to other columns in the current table are not allowed). The data type of the default expression must match the data type of the column.</p> <p>The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is NULL.</p> | - |

| Parameter | Description | Value Range or Example |
|--|--|---|
| UNIQUE [NULLS [NOT] DISTINCT NULLS IGNORE]
index_parameters

UNIQUE [NULLS [NOT] DISTINCT NULLS IGNORE]
(column_name [, ...])
index_parameters | <p>Specifies that a group of one or more columns of a table can contain only unique values.</p> <p>The [NULLS [NOT] DISTINCT NULLS IGNORE] field is used to specify how to process null values in the index column of the Unique index.</p> | <p>Default value: NULL can be inserted repeatedly.</p> <p>When the inserted data is compared with the original data in the table, the NULL value can be processed in any of the following ways:</p> <ul style="list-style-type: none">• NULLS DISTINCT: Each NULL value is regarded as a different value, that is, NULL values can be inserted repeatedly.• NULLS NOT DISTINCT: All NULL values are regarded as the same value. That is, only one NULL value entry is retained in the index. NULL values cannot be inserted repeatedly. Only distinct non-NULL values can be inserted.• NULLS IGNORE: NULL values are skipped during the equivalent comparison. If all index columns are NULL, NULL values can be inserted repeatedly. If some index columns are NULL, data can be inserted only when Not NULL values are different. <p>Table 21-154 lists the behaviors of the three processing modes.</p> <p>For details, see Application Differences Among NULLS DISTINCT, NULLS NOT DISTINCT, and NULLS IGNORE.</p> |

| Parameter | Description | Value Range or Example |
|---|---|--|
| | | CAUTION
If DISTRIBUTE BY REPLICATION is not specified, the column table that contains only unique values must contain distribution columns. |
| PRIMARY KEY
index_parameters
PRIMARY KEY
(column_name
[, ...])
index_parameters | <p>Specifies the primary key constraint specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.</p> <p>Only one primary key can be specified for a table.</p> <p>CAUTION
If DISTRIBUTE BY REPLICATION is not specified, the column set with a primary key constraint must contain distributed columns.</p> | - |
| DEFERRABLE
NOT DEFERRABLE | <p>Sets whether the constraint can be deferrable. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the SET CONSTRAINTS command. NOT DEFERRABLE is the default value. Currently, only UNIQUE and PRIMARY KEY constraints of row-store tables accept this clause. All the other constraints are not deferrable.</p> | - |

| Parameter | Description | Value Range or Example |
|---|--|------------------------|
| INITIALLY
IMMEDIATE
INITIALLY
DEFERRED | <p>If a constraint is deferrable, this clause specifies the default time to check the constraint.</p> <ul style="list-style-type: none">• If the constraint is INITIALLY IMMEDIATE (default value), it is checked after each statement.• If the constraint is INITIALLY DEFERRED, it is checked only at the end of the transaction. <p>The constraint check time can be altered using the SET CONSTRAINTS command.</p> | - |

table_constraint Parameters

Table constraints are similar to that of column constraints. For details, see [CONSTRAINT constraint_name Parameter](#).

```
{ CHECK ( expression ) |  
  UNIQUE [ NULLS [NOT] DISTINCT | NULLS IGNORE ] ( column_name [, ... ] ) index_parameters |  
  PRIMARY KEY ( column_name [, ... ] ) index_parameters }  
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

LIKE Parameters

like_option is as follows:

```
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS |  
  REOPTIONS | DISTRIBUTION | ALL }
```

The new table and original table are decoupled after creation is complete. Changes to the original table will not be applied to the new table, and it is not possible to include data of the new table in scans of the original table.

Default expressions for the copied column definitions will only be copied if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.

NOT NULL constraints are always copied to the new table. **CHECK** constraints will only be copied if **INCLUDING CONSTRAINTS** is specified; other types of constraints will never be copied. These rules also apply to column constraints and table constraints.

Columns and constraints copied by **LIKE** are not merged with the same name. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

- Any indexes on the source table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.

- STORAGE settings for the copied column definitions will only be copied if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- Comments for the copied columns, constraints, and indexes will only be copied if **INCLUDING COMMENTS** is specified. The default behavior is to exclude comments.
- If **INCLUDING REOPTIONS** is specified, the new table will copy the storage parameter (**WITH** clause of the source table) of the source table. The default behavior is to exclude partition definition of the storage parameter of the source table.
- If **INCLUDING DISTRIBUTION** is specified, the new table will copy the distribution information of the source table, including distribution type and column, and the new table cannot use **DISTRIBUTE BY** clause. The default behavior is to exclude distribution information of the source table.
- **INCLUDING ALL** is an abbreviated form of **INCLUDING DEFAULTS INCLUDING CONSTRAINTS INCLUDING INDEXES INCLUDING STORAGE INCLUDING COMMENTS INCLUDING REOPTIONS INCLUDING DISTRIBUTION**.

WITH Clause Parameters

Table 21-152 WITH parameters

| Parameter | Description | Value Range |
|------------|---|-------------|
| FILLFACTOR | The fillfactor of a table is a percentage between 10 and 100. 100 (complete packing) is the default value. When a smaller fillfactor is specified, INSERT operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose records are never updated, setting the fillfactor to 100 (complete packing) is the appropriate choice, but in heavily updated tables smaller fillfactors are appropriate. The parameter has no meaning for column-store tables. | 10~100 |

| Parameter | Description | Value Range |
|----------------------|---|---|
| ORIENTATION | Determines the storage mode of the data in the table.
CAUTION
orientation cannot be modified. | <ul style="list-style-type: none">• COLUMN: The data will be stored in columns.• ROW (default value): The data will be stored in rows.• ORC: The data of the table will be stored in ORC format (only HDFS table). |
| COMPRESSION | This parameter is available only for column-store tables. Value: LOW (default value), MIDDLE , or HIGH .
NOTE
Currently, row-store table compression is not supported. | - |
| MAX_BATCHROW | Specifies the maximum of a storage unit during data loading process. The parameter is only valid for column-store tables. | 10000 to 60000
Default value: 60000 |
| PARTIAL_CLUSTER_ROWS | Specifies the number of records to be partial cluster stored during data loading process. The parameter is only valid for column-store tables. | The value is no less than 100,000. The value is the multiple of MAX_BATCHROW . |
| enable_delta | Specifies whether to enable delta tables in column-store tables. The parameter is only valid for column-store tables. Using column-store tables with delta tables is not recommended. This may cause disk bloat and performance deterioration due to delayed merge. | Default value: off |
| DELTAROW_THRESHOLD | A reserved parameter. The parameter is only valid for column-store tables. | 0 to 60000
Default value: 6000 |

| Parameter | Description | Value Range |
|-----------------|--|----------------------------|
| COLD_TABLESPACE | Specifies the OBS tablespace for the cold partitions in a hot or cold table. This parameter is available only to column-store partitioned tables and cannot be modified. It must be used together with STORAGE_POLICY . The parameter STORAGE_POLICY can be left unconfigured. In this case, the default value default_obs_tbs is used. | Valid OBS tablespace name. |

| Parameter | Description | Value Range |
|----------------|--|--|
| STORAGE_POLICY | Specifies the rule for switching between hot and cold partitions. This parameter is used only for multi-temperature tables. This parameter must be used together with COLD_TABLESPACE . | <p>Value range: <i>Cold and hot switchover policy name</i>. <i>Cold and hot switchover threshold</i>. Currently, only LMT and HPN policies are supported. LMT indicates that the switchover is performed based on the last update time of partitions. HPN indicates the switchover is performed based on a fixed number of reserved hot partitions.</p> <ul style="list-style-type: none">• LMT:<i>[day]</i>: Switch the hot partition data that is not updated in the last <i>[day]</i> days to the OBS tablespace as cold partition data. <i>[day]</i> is an integer ranging from 0 to 36500, in days.• HPN:
<i>[hot_partition_num]</i>: <i>[hot_partition_num]</i> indicates the number of hot partitions (with data) to be retained. The rule is to find the maximum sequence ID of the partitions with data. The partitions without data whose sequence ID is greater than the maximum sequence ID are hot partitions, and <i>[hot_partition_num]</i> partitions are retained as hot partitions in descending order according to the sequence ID. A partition whose sequence ID is smaller than the minimum sequence ID of the |

| Parameter | Description | Value Range |
|---------------------------------|--|--|
| | | <p>retained hot partition is a cold partition. During hot and cold partition switchover, data needs to be migrated to the OBS tablespace.</p> <p><i>[hot_partition_num]</i> is an integer ranging from 0 to 1600.</p> <p>CAUTION</p> <ul style="list-style-type: none">For a LIST partition, you are advised to use the HPN policy with caution. Otherwise, the new partition may not be a hot partition. |
| ENABLE_MULTI_T
EMP_DISKCACHE | This parameter improves the performance of tables that separate cold and hot data. It is only applicable to tables that support this feature and is supported in versions 9.1.0 and later. | on or off
Default value: on/true |

| Parameter | Description | Value Range |
|-----------|--|--|
| PERIOD | <p>Specifies the period of automatically creating partitions and enables the automatic partition creation function. Only row-store and column-store range partitioned tables, time series tables, and cold and hot tables are supported. The partition key must be unique and its type can only be TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE] or DATE. MAXVALUE partitions are not supported. The value of (nowTime - boundaryTime)/ PERIOD must be less than the upper limit of the number of partitions, where nowTime indicates the current time and boundaryTime indicates the earliest partition boundary time. It cannot be used on midrange computers and acceleration clusters.</p> | <p>1 hour ~ 100 years</p> <p>CAUTION</p> <ul style="list-style-type: none">• In a database compatible with Teradata or MySQL, if the partition key type is DATE, PERIOD cannot be less than 1 day.• If PERIOD is set when a partitioned table is created, you can specify only the partition key. Two default partitions are created during table creation. The time ranges of the two default partitions are both PERIOD. The boundary time of the first default partition is the first hour, day, week, month, or year past the current time. The time unit is selected based on the maximum unit of PERIOD. The boundary time of the second default partition is the boundary time of the first partition plus PERIOD. Assume that the current time is 2022-02-17 16:32:45, and the boundary of the first default partition is described in Table 21-155. For more information about the default partitions, see Example: Creating a Partitioned Table with Automatic Partition Management and No Specified Partitions. |

| Parameter | Description | Value Range |
|-----------|---|--------------------|
| TTL | <p>Specifies the partition expiration time in partition management and enables the automatic partition deletion function. This parameter cannot be set separately. You must set PERIOD in advance or at the same time. The value of this parameter must be greater than or equal to that of PERIOD.</p> <p>Time To Live (TTL) specifies the data storage period of the table. The data that exceeds the TTL period will be cleared. To do this, the function in proc_drop_partition (relname regclass, older_than interval) is automatically invoked based on the period. The PERIOD and TTL values are of the Interval type, for example, 1 hour, 1 day, 1 week, 1 month, 1 year, and 1 month 2 day 3 hour.</p> <p>PERIOD indicates that data is partitioned by time period. The partition size may affect the query performance. The proc_add_partition (relname regclass, boundaries_interval interval) function is automatically invoked to create a partition after each period.</p> | 1 hour ~ 100 years |

| Parameter | Description | Value Range |
|---------------|--|---|
| COLVERSION | Specifies the version of the column-store format. Switching between different storage formats is supported. However, the storage format of a partitioned table cannot be switched. | <ul style="list-style-type: none">• 1.0: Each column in a column-store table is stored in a separate file. The file name is relfilenode.C1.0, relfilenode.C2.0, relfilenode.C3.0, or similar.• 2.0: All columns of a column-store table are combined and stored in a file. The file is named relfilenode.C1.0. Default value: 2.0
The value of COLVERSION can only be set to 2.0 for OBS multi-temperature tables.
CAUTION
When creating a column-store table, set COLVERSION to 2.0 . Compared with the 1.0 storage format, the performance is significantly improved: <ol style="list-style-type: none">1. The time required for creating a column-store wide table is significantly reduced.2. In the Roach data backup scenario, the backup time is significantly reduced.3. The build and catch up time is greatly reduced. |
| SKIP_FPI_HINT | Indicates whether to skip the hint bits operation when the full-page writes (FPW) log needs to be written during sequential scanning. | Default value: false
CAUTION
If SKIP_FPI_HINT is set to true and the checkpoint operation is performed on a table, no Xlog will be generated when the table is sequentially scanned. This applies to intermediate tables that are queried less frequently, reducing the size of Xlogs and improving query performance. |

PARTITION BY Parameters

- **partition_less_than_item** is as follows:
`PARTITION partition_name VALUES LESS THAN ( { partition_value | MAXVALUE } )`
- **partition_start_end_item** is as follows:
`PARTITION partition_name {
 {START(partition_value) END (partition_value) EVERY (interval_value)} |
 {START(partition_value) END ({partition_value | MAXVALUE})} |
 {START(partition_value)} |
 {END({partition_value | MAXVALUE})}
}`
- **list_partition_item**:
`PARTITION partition_name VALUES ( { (partition_value) [, ...] | DEFAULT } )`

Table 21-153 PARTITION BY Parameters

| Parameter | Description | Value Range |
|-----------------------------------|---|--|
| PARTITION BY RANGE(partition_key) | The syntax specifying range partitioning. partition_key indicates the name of a partition key. | <ul style="list-style-type: none">Assume that the VALUES LESS THAN syntax is used.
CAUTION
In this case, a maximum of four partition keys are supported, and the partition key must be a column name. If there are multiple partition keys, a column name can appear only once, and two adjacent partition keys must be separated by a comma (,).

Data types supported by the partition keys are as follows:
SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, REAL, DOUBLE PRECISION, CHARACTER VARYING(n), VARCHAR(n), CHARACTER(n), CHAR(n), CHARACTER, CHAR, TEXT, NVARCHAR2, NAME, TIMESTAMP[(p)] [WITHOUT TIME ZONE], TIMESTAMP[(p)] [WITH TIME ZONE], and DATE.Assume that the START END syntax is used.
CAUTION
In this case, only one partition key is supported.

Data types supported by the partition key are as follows: SMALLINT, INTEGER, BIGINT, DECIMAL, NUMERIC, REAL, DOUBLE PRECISION, TIMESTAMP[(p)] [WITHOUT TIME |

| Parameter | Description | Value Range |
|---|--|---|
| | | ZONE],
TIMESTAMP[(p)]
[WITH TIME ZONE],
and DATE. |
| PARTITION BY
LIST
(partition_key,...) | The syntax specifying list partitioning. partition_key indicates the name of a partition key. | In list partitioning, partition keys support the following data types: TINYINT, SMALLINT, INTEGER, BIGINT, NUMERIC/DECIMAL, TEXT, NVARCHAR2, VARCHAR(n), CHAR, BPCHAR, TIME, TIME WITH TIMEZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, DATE, INTERVAL and SMALLDATETIME.
CAUTION
In list partitioning, a partition key has a maximum of four columns. |

| Parameter | Description | Value Range |
|--------------------------|---|-------------|
| partition_less_than_item | <p>Partition definition syntax in range partitioning.</p> <p>partition_name is the name of a range partition.</p> <p>partition_value is the upper limit of range partition, and the value depends on the type of partition_key. MAXVALUE can specify the upper boundary of a range partition, and it is commonly used to specify the upper boundary of the last range partition.</p> <pre>PARTITION partition_name VALUES
LESS THAN ({ partition_value
MAXVALUE })</pre> <p>CAUTION</p> <ul style="list-style-type: none">• Upper boundaries must be specified for each partition.• The types of upper boundaries must be the same as those of partition keys.• In a partition list, partitions are arranged in ascending order of upper boundary values. Therefore, a partition with a certain upper boundary value is placed before another partition with a larger upper boundary value.• If a partition key consists of multiple columns, the columns are used for partitioning in sequence. The first column is preferred to be used for partitioning. If the values of the first columns are the same, the second column is used. The subsequent columns are used in the same manner. | - |

| Parameter | Description | Value Range |
|--------------------------|--|--|
| partition_start_end_item | <p>The syntax of using the start value and interval value to define a range partition.</p> <pre>PARTITION partition_name {START (partition_value) END (partition_value) EVERY (interval_value)}
 {START (partition_value) END (partition_value) MAXVALUE}

{START(partition_value)
 {END (partition_value) MAXVALUE}}</pre> | <p>The following table describes the parameters.</p> <ul style="list-style-type: none">• partition_name: name or name prefix of a range partition. It is the name prefix only in the following cases (assuming that partition_name is p1):<ul style="list-style-type: none">– If START+END+EVERY is used, the names of partitions will be defined as p1_1, p1_2, and the like. For example, if PARTITION p1 START(1) END(4) EVERY(1) is defined, the generated partitions are [1, 2), [2, 3), and [3, 4), and their names are p1_1, p1_2, and p1_3. In this case, p1 is a name prefix.– If the defined statement is in the first place and has START specified, the range (MINVALUE, START) will be automatically used as the first actual partition, and its name will be p1_0. The other partitions are then named p1_1, p1_2, and the like. For example, if PARTITION p1 START(1), PARTITION p2 START(2) is defined, generated partitions are (MINVALUE, 1), [1, 2), and [2, MAXVALUE), and their names will be p1_0, p1_1, and p2. |

| Parameter | Description | Value Range |
|-----------|-------------|--|
| | | <p>In this case, p1 is a name prefix and p2 is a partition name. MINVALUE means the minimum value.</p> <ul style="list-style-type: none">• partition_value: start point value or end point value of a range partition. The value depends on partition_key and cannot be MAXVALUE.• interval_value: width of each partition for dividing the [START, END) range. It cannot be MAXVALUE. If the value of (END – START) divided by EVERY has a remainder, the width of only the last partition is less than the value of EVERY.• MAXVALUE: maximum value. It is usually used to set the upper boundary for the last range partition. |

| Parameter | Description | Value Range |
|-----------|-------------|--|
| | | <p>CAUTION</p> <ol style="list-style-type: none">If the defined statement is in the first place and has START specified, the range (MINVALUE, START) will be automatically used as the first actual partition.The START END syntax must comply with the following rules:<ul style="list-style-type: none">The value of START (if any, same for the following situations) in each partition_start_end_item must be smaller than that of END.In two adjacent partition_start_end_item statements, the value of the first END must be equal to that of the second START.The value of EVERY in each partition_start_end_item must be a positive number (in ascending order) and must be smaller than END minus START.Each partition includes the start value (unless it is MINVALUE) and excludes the end value. The format is as follows: [Start value, end value).Partitions created by the same partition_start_end_item belong to the same tablespace.If partition_name is a name prefix of a partition, the length must not exceed 57 bytes. If there are more than 57 bytes, the prefix will be automatically truncated.When creating or modifying a partitioned table, ensure that the total number of partitions in the table does not exceed the maximum value (32767). <ol style="list-style-type: none">In statements for creating partitioned |

| Parameter | Description | Value Range |
|---------------------|---|--|
| list_partition_item | <p>Partition definition syntax in list partitioning.</p> <p>partition_name indicates the partition name.</p> <p>partition_value is an enumerated value of the list partition boundary. The value depends on the type of partition_key. DEFAULT indicates the default partition boundary.</p> <p><code>PARTITION partition_name VALUES ({ (partition_value) [, ...] DEFAULT })</code></p> | <p>The following conventions and constraints apply to list partitioned tables:</p> <ul style="list-style-type: none">• The partition whose boundary value is DEFAULT is the default partition.• Each list partitioned table can have only one DEFAULT partition.• The number of partitions in a partitioned table cannot exceed 32767, and the number of boundary values of all partitions cannot exceed 32767.• Regardless of the number of partition keys, the boundary of the DEFAULT partition can only be DEFAULT.• If a partition key consists of multiple columns, each partition_value must contain the values of all partition keys. If a partition key contains only one column, the parentheses on both sides of partition_value can be omitted. For details, see Example: Creating a List Partitioned Table with Multiple Partition Keys.• If the partition key consists of multiple columns, compare the values in the columns one by one. If a value is different from another value, regardless of their columns, they are different values. |

| Parameter | Description | Value Range |
|-----------|-------------|---|
| | | <ul style="list-style-type: none">Each value of partition_value must be unique.When data is inserted, if its partition key and value falls into the boundary of a non-DEFAULT partition, the data is written to the partition. Otherwise, the data is written to the DEFAULT partition. |

Table 21-154 Processing of NULL values in index columns in unique indexes

| Constraint | All Index Columns Are NULL | Some Index Columns Are NULL |
|--------------------|--------------------------------|--|
| NULLS DISTINCT | Can be inserted repeatedly. | Can be inserted repeatedly. |
| NULLS NOT DISTINCT | Cannot be inserted repeatedly. | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |
| NULLS IGNORE | Can be inserted repeatedly. | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |

Table 21-155 Partition boundaries

| period | Maximum PERIOD Unit | Boundary of First Default Partition |
|---------|---------------------|-------------------------------------|
| 1hour | Hour | 2022-02-17 17:00:00 |
| 1day | Day | 2022-02-18 00:00:00 |
| 1month | Month | 2022-03-01 00:00:00 |
| 13month | Year | 2023-01-01 00:00:00 |

DISTRIBUTE BY Parameter Description

Default value: determined by the GUC parameter **default_distribution_mode**

- When **default_distribution_mode** is set to **roundrobin**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 - a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - b. If the primary key or unique constraint is not included during table creation, round-robin distribution is selected.
- When **default_distribution_mode** is set to **hash**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 - a. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - b. 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 columns, hash distribution is selected. The distribution column is the first column whose data type can be used as a distribution column.
 - c. 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 column exists, round-robin distribution is selected.

The following data types can be used as distribution columns:

- Integer types: **TINYINT**, **SMALLINT**, **INT**, **BIGINT**, and **NUMERIC/DECIMAL**
- Character types: **CHAR**, **BPCHAR**, **VARCHAR**, **VARCHAR2**, **NVARCHAR2**, and **TEXT**
- Date/time types: **DATE**, **TIME**, **TIMETZ**, **TIMESTAMP**, **TIMESTAMPTZ**, **INTERVAL**, and **SMALLDATETIME**

Example: Using the LESS THAN Syntax to Create a Range Partitioned Table

The range partitioned table **customer_address** has four partitions and their partition keys are of the integer type. The ranges of the partitions are as follows: **ca_address_sk < 2450815**, **2450815 <= ca_address_sk < 2451179**, **2451179 <= ca_address_sk < 2451544**, **2451544 <= ca_address_sk**.

```
CREATE TABLE 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)
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE(ca_address_sk)
(
    PARTITION P1 VALUES LESS THAN(2450815),
    PARTITION P2 VALUES LESS THAN(2451179),
    PARTITION P3 VALUES LESS THAN(2451544),
    PARTITION P4 VALUES LESS THAN(MAXVALUE)
);
```

View the information of the partitioned table.

```
SELECT relname, boundaries FROM pg_partition p where p.parentid='customer_address'::regclass ORDER BY 1;
```

| relname | boundaries |
|------------------|------------|
| customer_address | |
| p1 | {2450815} |
| p2 | {2451179} |
| p3 | {2451544} |
| p4 | {NULL} |
| (5 rows) | |

Query the number of rows in the **P1** partition.

```
SELECT count(*) FROM customer_address PARTITION (P1);
SELECT count(*) FROM customer_address PARTITION FOR (2450815);
```

Examples: Enabling and Disabling ROW MOVEMENT

If **ROW MOVEMENT** is enabled, the data of a row is moved to a new partition when the partition key of the row is updated to a new value and the new value belongs to another partition.

If a partition key is frequently updated, enabling **ROW MOVEMENT** may cause extra I/O overhead. To enable it, evaluate service requirements first.

Create a table partitioned by date and enable **ROW MOVEMENT**.

```
DROP TABLE IF EXISTS sale_data;
CREATE TABLE sales_data (
  sale_id INT,
  product_name VARCHAR(100),
  sale_date DATE,
  amount DECIMAL(10, 2)
)
PARTITION BY RANGE (sale_date) (
  PARTITION p_2023q1 VALUES LESS THAN ('2023-04-01'),
  PARTITION p_2023q2 VALUES LESS THAN ('2023-07-01'),
  PARTITION p_2023q3 VALUES LESS THAN ('2023-10-01'),
  PARTITION p_2023q4 VALUES LESS THAN ('2024-01-01')
)
ENABLE ROW MOVEMENT;
```

Insert test data.

```
INSERT INTO sales_data (sale_id, product_name, sale_date, amount)
VALUES
(1, 'Product A', '2023-02-15', 1000.00), -- It belongs to p_2023q1.
(2, 'Product B', '2023-05-20', 1500.00); -- It belongs to p_2023q2.
```

Query the data distribution of each partition.

```
SELECT tableoid::regclass AS partition, * FROM sales_data ORDER BY sale_id;
```

| partition | sale_id | product_name | sale_date | amount |
|------------|---------|--------------|---------------------|---------|
| 2147617615 | 1 | Product A | 2023-02-15 00:00:00 | 1000.00 |
| 2147617616 | 2 | Product B | 2023-05-20 00:00:00 | 1500.00 |
| (2 rows) | | | | |

Update the partition key (which is moved from **p_2023q1** to **p_2023q2**).

```
UPDATE sales_data SET sale_date = '2023-05-01' WHERE sale_id = 1;
```

Query the data distribution of each partition again to confirm that the row has been moved.

```
SELECT tableoid::regclass AS partition, * FROM sales_data ORDER BY sale_id;
```

| partition | sale_id | product_name | sale_date | amount |
|------------|---------|--------------|---------------------|---------|
| 2147617616 | 1 | Product A | 2023-05-01 00:00:00 | 1000.00 |
| 2147617616 | 2 | Product B | 2023-05-20 00:00:00 | 1500.00 |

(2 rows)

Disable **ROW MOVEMENT**.

```
ALTER TABLE sales_data DISABLE ROW MOVEMENT;
```

Modify the partition key again (which is move from **p_2023q2** back to **p_2023q2**).

```
UPDATE sales_data SET sale_date = '2023-02-15' WHERE sale_id = 1;
```

View the command output. It indicates that **ROW MOVEMENT** is disabled and the partition key cannot be updated.

```
gaussdb=> UPDATE sales_data SET sale_date = '2023-02-15' WHERE sale_id = 1;
ERROR:  dn_6003_6004: fail to update partitioned table "sales_data"
DETAIL:  disable row movement
```

Example: Using the START END Syntax to Create a Column-store Range Partitioned Table

```
CREATE TABLE customer_address_SE
(
    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)
)
WITH (ORIENTATION = COLUMN)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE(ca_address_sk)
(
    PARTITION p1 START(1) END(1000) EVERY(200),
    PARTITION p2 END(2000),
    PARTITION p3 START(2000) END(5000)
);
```

View the information of the partitioned table.

```
SELECT relname, boundaries FROM pg_partition p where p.parentid='customer_address_SE'::regclass ORDER
BY 1;
 relname | boundaries
-----+-----
customer_address_se |
p1_0      | {1}
p1_1      | {201}
p1_2      | {401}
p1_3      | {601}
p1_4      | {801}
p1_5      | {1000}
p2        | {2000}
p3        | {5000}
(9 rows)
```

Example: Using the START END Syntax to Create and Modify a Range Partitioned Table

Create a temporary schema.

```
CREATE SCHEMA tpcds;  
SET CURRENT_SCHEMA TO tpcds;
```

Create a partitioned table with the partition key of the integer type.

```
CREATE TABLE tpcds.startend_pt (c1 INT, c2 INT)  
DISTRIBUTE BY HASH (c1)  
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)  
)  
ENABLE ROW MOVEMENT;
```

View the information of the partitioned table.

```
SELECT relname, boundaries FROM pg_partition p where p.parentid='tpcds.startend_pt'::regclass ORDER BY  
1;
```

| relname | boundaries |
|-------------------|------------|
| p1_0 | {1} |
| p1_1 | {201} |
| p1_2 | {401} |
| p1_3 | {601} |
| p1_4 | {801} |
| p1_5 | {1000} |
| p2 | {2000} |
| p3 | {2500} |
| p4 | {3000} |
| p5_1 | {4000} |
| p5_2 | {5000} |
| tpcds.startend_pt | |

(12 rows)

Import data and check the data volume in the partition.

```
INSERT INTO tpcds.startend_pt VALUES (GENERATE_SERIES(0, 4999), GENERATE_SERIES(0, 4999));  
SELECT COUNT(*) FROM tpcds.startend_pt PARTITION FOR (0);
```

| count |
|-------|
| 1 |

(1 row)

```
SELECT COUNT(*) FROM tpcds.startend_pt PARTITION (p3);
```

| count |
|-------|
| 500 |

(1 row)

View the information of the partitioned table.

```
SELECT relname, boundaries FROM pg_partition p where p.parentid='tpcds.startend_pt'::regclass ORDER BY  
1;
```

| relname | boundaries |
|---------|------------|
| p1_0 | {1} |
| p1_1 | {201} |
| p1_2 | {401} |
| p1_3 | {601} |
| p1_4 | {801} |
| p1_5 | {1000} |
| p2 | {2000} |
| p3 | {2500} |
| p4 | {3000} |
| p5_1 | {4000} |
| p6_1 | {5300} |
| p6_2 | {5600} |
| p6_3 | {5900} |

```
p71      | {6000}  
q1_1     | {4250}  
q1_2     | {4500}  
q1_3     | {4750}  
q1_4     | {5000}  
tpcds.startend_pt |  
(19 rows)
```

Example: Creating a List Partitioned Table with Partition Keys

```
CREATE TABLE data_list  
(  
    id int,  
    time int,  
    salary decimal(12,2)  
)  
PARTITION BY LIST (time)  
(  
    PARTITION P1 VALUES (202209),  
    PARTITION P2 VALUES (202210,202208),  
    PARTITION P3 VALUES (202211),  
    PARTITION P4 VALUES (202212),  
    PARTITION P5 VALUES (202301)  
);
```

Example: Creating a List Partitioned Table with Multiple Partition Keys

A partitioned table has two partition keys: **period** and **city**.

```
CREATE TABLE sales_info  
(  
    sale_time timestamptz,  
    period    int,  
    city      text,  
    price     numeric(10,2),  
    remark    varchar2(100)  
)  
DISTRIBUTE BY HASH(sale_time)  
PARTITION BY LIST (period, city)  
(  
    PARTITION north_2022 VALUES (('202201', 'north1'), ('202202', 'north2')),  
    PARTITION south_2022 VALUES (('202201', 'south1'), ('202202', 'south2'), ('202203', 'south2')),  
    PARTITION rest VALUES (DEFAULT)  
);
```

Example: Creating a Partitioned Table with Automatic Partition Management and No Specified Partitions

Set **PERIOD** to **1 day** and partition key to **time**.

```
CREATE TABLE time_part  
(  
    id integer,  
    time timestamp  
) with (PERIOD='1 day')  
partition by range(time);
```

Two default partitions are created during table creation. The boundary time of the first default partition is the start of the day later than the current time, that is, 2022-12-13 00:00:00. The boundary time of the second default partition is the boundary time of the first partition plus PERIOD, that is, 2022-12-13 00:00:00+1 day=2022-12-14 00:00:00.

```
SELECT now();  
       now  
-----
```



```
2022-12-12 20:41:21.603172+08
(1 row)

SELECT relname, boundaries FROM pg_partition p where p.parentid='time_part'::regclass ORDER BY 1;
 relname | boundaries
-----+-----
default_part_1 | {"2022-12-13 00:00:00"}
default_part_2 | {"2022-12-14 00:00:00"}
time_part |
(3 rows)
```

Example: Creating a Partitioned Table with Automatic Partition Management and Specified Partitions

```
CREATE TABLE CPU(
  id integer,
  idle numeric,
  IO numeric,
  scope text,
  IP text,
  time timestamp
) with (TTL='7 days',PERIOD='1 day')
partition by range(time)
(
  PARTITION P1 VALUES LESS THAN('2022-01-05 16:32:45'),
  PARTITION P2 VALUES LESS THAN('2022-01-06 16:56:12')
);
```

Example: Creating a Partitioned Table by Month

Create partitioned table **customer_address**, which contains 13 partitions and their partition keys are of the date type.

```
CREATE TABLE customer_address
(
  ca_address_sk integer NOT NULL,
  ca_address_date date NOT NULL
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE (ca_address_date)
(
  PARTITION p202001 VALUES LESS THAN('20200101'),
  PARTITION p202002 VALUES LESS THAN('20200201'),
  PARTITION p202003 VALUES LESS THAN('20200301'),
  PARTITION p202004 VALUES LESS THAN('20200401'),
  PARTITION p202005 VALUES LESS THAN('20200501'),
  PARTITION p202006 VALUES LESS THAN('20200601'),
  PARTITION p202007 VALUES LESS THAN('20200701'),
  PARTITION p202008 VALUES LESS THAN('20200801'),
  PARTITION p202009 VALUES LESS THAN('20200901'),
  PARTITION p202010 VALUES LESS THAN('20201001'),
  PARTITION p202011 VALUES LESS THAN('20201101'),
  PARTITION p202012 VALUES LESS THAN('20201201'),
  PARTITION p202013 VALUES LESS THAN(MAXVALUE)
);
```

Insert data.

```
INSERT INTO customer_address values('1','20200215');
INSERT INTO customer_address values('7','20200805');
INSERT INTO customer_address values('9','20201111');
INSERT INTO customer_address values('4','20201231');
```

Query a partition.

```
SELECT * FROM customer_address PARTITION(p202009);
ca_address_sk | ca_address_date
-----+-----
```

```
7 | 2020-08-05 00:00:00  
(1 row)
```

Example: Using the START END Syntax to Create a Partitioned Table with Multiple Partitions at a Time

- Create a partitioned table **day_part**. Each day is a partition, and the partition key is a date.

```
CREATE table day_part(id int,d_time date)  
DISTRIBUTE BY HASH (id)  
PARTITION BY RANGE (d_time)  
(PARTITION p1 START('2022-01-01') END('2022-01-31') EVERY(interval '1 day'));  
ALTER TABLE day_part ADD PARTITION pmax VALUES LESS THAN (maxvalue);
```

- Create a partitioned table **week_part**, seven days as a partition, and the partition key is a date.

```
CREATE table week_part(id int,w_time date)  
DISTRIBUTE BY HASH (id)  
PARTITION BY RANGE (w_time)  
(PARTITION p1 START('2021-01-01') END('2022-01-01') EVERY(interval '7 day'));  
ALTER TABLE week_part ADD PARTITION pmax VALUES LESS THAN (maxvalue);
```

- Create the partitioned table **month_part**, each month as a partition, and the partition key is a date.

```
CREATE table month_part(id int,m_time date)  
DISTRIBUTE BY HASH (id)  
PARTITION BY RANGE (m_time)  
(PARTITION p1 START('2021-01-01') END('2022-01-01') EVERY(interval '1 month'));  
ALTER TABLE month_part ADD PARTITION pmax VALUES LESS THAN (maxvalue);
```

Example: Creating a Hot or Cold Table

Only a column-store partitioned table is supported. Use the default OBS tablespace. Set LMT to 30 for cold and hot switchover rules.

```
CREATE TABLE cold_hot_table  
(  
  W_WAREHOUSE_ID      CHAR(16)      NOT NULL,  
  W_WAREHOUSE_NAME     VARCHAR(20)    ,  
  W_STREET_NUMBER      CHAR(10)      ,  
  W_STREET_NAME        VARCHAR(60)   ,  
  W_STREET_ID          CHAR(15)      ,  
  W_SUITE_NUMBER       CHAR(10)      ,  
)  
WITH (ORIENTATION = COLUMN, storage_policy = 'LMT:30')  
DISTRIBUTE BY HASH (W_WAREHOUSE_ID)  
PARTITION BY RANGE(W_STREET_ID)  
(  
  PARTITION P1 VALUES LESS THAN(100000),  
  PARTITION P2 VALUES LESS THAN(200000),  
  PARTITION P3 VALUES LESS THAN(300000),  
  PARTITION P4 VALUES LESS THAN(MAXVALUE)  
)  
ENABLE ROW MOVEMENT;
```

Helpful Links

[ALTER TABLE PARTITION, DROP TABLE](#)

21.13.4.3 CREATE TABLE AS

Function

CREATE TABLE AS creates a table based on the results of a query. It is applicable to data migration or replication scenarios.

CREATE TABLE AS creates a table and fills the table with **SELECT** query results. The table columns have the names and data types associated with the output columns of **SELECT**. Except that you can override the **SELECT** output column names by giving an explicit list of new column names.

CREATE TABLE AS queries once the source table and writes data in the new table. The query result view changes when the source table changes. In contrast, a view re-evaluates its defining **SELECT** statement whenever it is queried.

Precautions

- This command cannot be used to create a partitioned table.
- If an error occurs when you create a table, after the system is recovered, the system probably cannot automatically clear the created disk file whose size is not 0. This problem seldom occurs.
- The UNLOGGED table uses no primary/standby mechanism. In the case of system faults or abnormal breakpoints, data loss may occur. Therefore, the UNLOGGED table cannot be used to store basic data.
- The UNLOGGED table does not ensure safe data. Users can back up data before using the table; for example, users should back up the data before a system upgrade.

Syntax

```
CREATE [ [ GLOBAL | LOCAL | VOLATILE ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE table_name
    [ ( column_name [, ...] ) ]
    [ WITH ( {storage_parameter = value} [, ...] ) ]
    [ COMPRESS | NOCOMPRESS ]

    [ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { [HASH] ( column_name ) } } ]

    [ COMMENT [=] 'text' ]
    AS query
    [ WITH [ NO ] DATA ];
```

Parameter Description

Table 21-156 CREATE TABLE AS parameters

| Parameter | Description | Value Range |
|---------------------------|--|--|
| GLOBAL LOCAL VOLATILE | Specifies the type of a temporary table. | For details, see Parameter Description . |
| TEMPORARY TEMP | If TEMP or TEMPORARY is specified, the created table is a temporary table. | - |

| Parameter | Description | Value Range |
|--|--|--|
| UNLOGGED | <p>Specifies that the table is created as an unlogged table. Data written to unlogged tables is not written to the write-ahead log, which makes them considerably faster than ordinary tables. However, they are not crash-safe: an unlogged table is automatically truncated after a crash or unclean shutdown. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.</p> <p>If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should re-create the indexes with errors.</p> | <p>Starting from version 9.1.0, UNLOGGED tables are automatically saved in the pg_unlogged tablespace and cannot be moved or assigned to other tablespaces. After an earlier version is upgraded to 9.1.0, the UNLOGGED table created in the earlier version is still stored in the original tablespace.</p> <p>If the instance restarts unexpectedly, the UNLOGGED table will be reset, which can impact the instance's recovery time objective (RTO). Starting from version 9.1.0, there is a script called switch_unlogged_tablespace.py that can move unlogged tables to optimize the recovery time objective (RTO). This script works together with the GUC parameter enable_unlogged_tablepace_compat.</p> |
| table_name | Specifies the name of the table to be created. | A string, which must comply with the naming convention . |
| column_name | Specifies the name of a column to be created in the new table. | A string, which must comply with the naming convention . |
| WITH
(storage_parameter [= value] [, ...]) | The WITH clause sets the storage options for a table or its index. | For details about the parameters, see Table 21-157 . |

| Parameter | Description | Value Range |
|--------------------------|--|-------------|
| COMPRESS
NOCOMPRESS | <ul style="list-style-type: none">• If the keyword COMPRESS is specified, the compression feature is triggered when data is inserted into tables in batches. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored.• If NOCOMPRESS is specified, the table is not compressed. Default value:
NOCOMPRESS , tuple data is not compressed before storage. | - |

| Parameter | Description | Value Range |
|--------------------|--|---|
| DISTRIBUTE BY | <p>Specifies how the table is distributed or replicated between DNs.</p> <ul style="list-style-type: none">• REPLICATION: Each row in the table exists on all DNs, that is, each DN has complete table data.• ROUNDROBIN: Each row in the table is sent to each DN in sequence. This distribution policy prevents data skew. However, data distribution nodes are random. As a result, there is a higher probability that table redistribution is triggered during computing. This distribution policy is recommended for large tables with severe column skew. This value is supported only in 8.1.2 or later.• HASH (column_name): Each row of the table will be placed into all the DNs based on the hash value of the specified column. <p>Default value: determined by the GUC parameter default_distribution_mode. For details about how to select the default value of DISTRIBUTE BY, see the notice below this table.</p> | <p>The following data types can be used as distribution columns:</p> <ul style="list-style-type: none">• Integer types: TINYINT, SMALLINT, INT, BIGINT, and NUMERIC/DECIMAL• Character types: CHAR, BPCHAR, VARCHAR, VARCHAR2, NVARCHAR2, and TEXT• Date/time types: DATE, TIME, TIMETZ, TIMESTAMP, TIMESTAMPPTZ, INTERVAL, and SMALLDATETIME |
| COMMENT [=] 'text' | The COMMENT clause can specify table comments during table creation. | - |

| Parameter | Description | Value Range |
|----------------------|---|-------------|
| AS query | Indicates a SELECT or VALUES command, or an EXECUTE command that runs a prepared SELECT , or VALUES query. | - |
| [WITH [NO] DATA] | Specifies whether the data produced by the query should be copied into the new table. By default, the data is copied. If the NO parameter is used, the data is not copied. | - |

NOTICE

DISTRIBUTE BY parameter description:

- When **DISTRIBUTE BY HASH (column_name)** is specified, the primary key and its unique index must contain the **column_name** column.
 - When **DISTRIBUTE BY HASH (column_name)** in a referenced table is specified, the foreign key of the reference table must contain the **column_name** column.
 - When **default_distribution_mode** is set to **roundrobin**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 1. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 2. If the primary key or unique constraint is not included during table creation, round-robin distribution is selected.
 - When **default_distribution_mode** is set to **hash**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 1. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 2. 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 columns, hash distribution is selected. The distribution column is the first column whose data type can be used as a distribution column.
 3. 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 column exists, round-robin distribution is selected.
-

Table 21-157 storage_parameter parameters

| storage_parameter
Parameter | Description | Value Range |
|--------------------------------|--|---|
| FILLFACTOR | <p>The fillfactor of a table is a percentage between 10 and 100. When a smaller fillfactor is specified, INSERT operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose records are never updated, setting the fillfactor to 100 (complete packing) is the appropriate choice, but in heavily updated tables smaller fillfactors are appropriate.</p> <p>The default value is 100 (filled to capacity).</p> <p>The parameter is only valid for row-store tables.</p> | 10~100 |
| ORIENTATION | <p>Specifies the storage mode (row-store, column-store) for table data. This parameter cannot be modified once it is set.</p> <p>Default value: ROW (row-store)</p> | <ul style="list-style-type: none">• ROW indicates that table data is stored in rows. ROW applies to OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using ROW can improve the efficiency.• COLUMN indicates that the data is stored in columns. COLUMN applies to the data warehouse service, which has a large amount of aggregation computing, and involves a few column operations. |

| storage_parameter
Parameter | Description | Value Range |
|--------------------------------|---|--|
| COMPRESSION | <p>Specifies the compression level of the table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time, and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution characteristics of loading table data.</p> <p>NOTE
Currently, row-store table compression is not supported.</p> | <p>This parameter is available only for column-store tables. Value: LOW (default value), MIDDLE, or HIGH.</p> |
| MAX_BATCHROW | <p>Specifies the maximum of a storage unit during data loading process. The parameter is only valid for column-store tables.</p> <p>Default value: 60000</p> | 10000 to 60000 |
| PARTIAL_CLUSTER_ROWS | <p>Specifies the number of records to be partial cluster stored during data loading process. The parameter is only valid for column-store tables.</p> <p>Default value: 4,200,000</p> | 600000 to 2147483647 |
| enable_delta | <p>Specifies whether to enable delta tables in column-store tables. The parameter is only valid for column-store tables.</p> <p>Using column-store tables with delta tables is not recommended. This may cause disk bloat and performance deterioration due to delayed merge.</p> <p>Default value: off</p> | on or off |

| storage_parameter
Parameter | Description | Value Range |
|--------------------------------|---|---|
| COLVERSION | <p>Specifies the version of the column-store format. You can switch between different storage formats.</p> <p>Default value: 2.0</p> <p>NOTE</p> <p>When creating a column-store table, set COLVERSION to 2.0. Compared with the 1.0 storage format, the performance is significantly improved:</p> <ol style="list-style-type: none">1. The time required for creating a column-store wide table is significantly reduced.2. In the Roach data backup scenario, the backup time is significantly reduced.3. The build and catch up time is greatly reduced. | <p>Valid value:</p> <ul style="list-style-type: none">• 1.0: Each column in a column-store table is stored in a separate file. The file name is relfilenode.C1.0, relfilenode.C2.0, relfilenode.C3.0, or similar.• 2.0: All columns of a column-store table are combined and stored in a file. The file is named relfilenode.C1.0. |
| SKIP_FPI_HINT | <p>Indicates whether to skip the hint bits operation when the full-page writes (FPW) log needs to be written during sequential scanning.</p> <p>Default value: false</p> <p>NOTE</p> <p>If SKIP_FPI_HINT is set to true and the checkpoint operation is performed on a table, no Xlog will be generated when the table is sequentially scanned. This applies to intermediate tables that are queried less frequently, reducing the size of Xlogs and improving query performance.</p> | true or false |

Examples

Create the **customer** table.

```
DROP TABLE IF EXISTS customer;
CREATE TABLE customer
(
  C_CUSTKEY    BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY ,
  C_NAME      VARCHAR(25) ,
  C_ADDRESS   VARCHAR(40) ,
  C_NATIONKEY INT NOT NULL CHECK (C_NATIONKEY > 0)
```

```
)  
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Create the **store_returns_t1** table and insert numbers that are greater than 4795 in the **C_CUSTKEY** column of the **CUSTOMER** table:

```
CREATE TABLE store_returns_t1 AS SELECT * FROM CUSTOMER WHERE C_CUSTKEY > 4795;
```

Copy **store_returns_t1** to create the **store_returns_t2** table:

```
CREATE TABLE store_returns_t2 AS table store_returns_t1;
```

Helpful Links

[CREATE TABLE, SELECT](#)

21.13.4.4 ALTER TABLE

Function

ALTER TABLE is used to modify tables, including modifying table definitions, renaming tables, renaming specified columns in tables, renaming table constraints, setting table schemas, enabling or disabling row-level access control, and adding or updating multiple columns.

Only the owner of a table, a user granted with the ALTER permission for the table, or a system administrator has the permission to run the **ALTER TABLE** statement. To change the owner or schema of a table, you must be the table owner or a system administrator and be a direct or indirect member of the new role.

Precautions

Table 21-158 Precautions for ALTER TABLE

| Constraint | Description |
|--|--|
| Storage method of tables | After a table is created, the parameter ORIENTATION (which specifies whether the table is a row-store or column-store table) cannot be modified. |
| Supported range of column-store tables | Column-store tables support PARTIAL CLUSTER KEY but do not support table-level foreign key constraints. In 8.1.1 or later, you can use the PRIMARY KEY constraint and table-level UNIQUE constraint during the column-store table creation. |
| Modification of column-store tables | <ul style="list-style-type: none">The NOT NULL constraint and PRIMARY KEY constraint can be added to existing column-store tables using ALTER. This constraint is supported only by clusters of version 8.2.0 or later.If you delete the PRIMARY KEY constraint by specifying the constraint name, the NOT NULL constraint is not deleted. You can manually delete the NOT NULL constraint as needed. |

| Constraint | Description |
|---|--|
| Columns of column-store tables | The column constraints supported by column-store tables include NULL , NOT NULL , and DEFAULT constant values. Only the DEFAULT value can be modified and deleted (SET DEFAULT and DROP DEFAULT), and only the NOT NULL constraint can be deleted. |
| Operations supported by column-store tables | <ul style="list-style-type: none">• ADD COLUMN is used to add a column. The types of new and modified columns should be the data types supported by column storage (for details, see Data Types).• ALTER TYPE is used to change the data type of a column. The USING option of ALTER TYPE supports only constant expressions and expressions involving the column.• SET STATISTICS is used to set the statistics collection target of a single column.• Table name can be modification.• DROP COLUMN is used to delete a column. |
| Modification of distribution columns | The data type of the distribution column in an existing table cannot be modified. |
| Row-store REPLICATION tables | A system column cannot be designated as a primary key in a row-store REPLICATION distributed table. |
| Auto-increment columns | Auto-increment columns cannot be added, or a column in which the DEFAULT value contains the nextval() expression cannot be added either. |
| Modification of OBS cold and hot tables | <ul style="list-style-type: none">• The cold_tablespace and storage_policy parameters of ALTER RESET cannot be used in OBS hot or cold tables, and COLVERSION cannot be changed to 1.0 for such tables.• You can change a column-store table whose COLVERSION is 2.0 to an OBS hot or cold table. The COLD_TABLESPACE and STORAGE_POLICY parameters must be added.• You can use ALTER TABLE to change the values of STORAGE_POLICY for REOPTIONS. After the cold/hot switchover policy is changed, the cold/hot attribute of the existing cold data will not change. The new policy takes effect for the next cold/hot switchover. |
| V3 tables | COLVERSION cannot be modified for V3 tables (COLVERSION is 3.0), and non-V3 tables (COLVERSION is 2.0) cannot be switched to V3 tables. |

| Constraint | Description |
|--|---|
| Operations that trigger table rebuilding | <p>When an ALTER TABLE operation is performed on a table, it triggers table rebuilding. During this process, data is dumped into a new data file. Once the process is complete, the original file is deleted. If the table is large, the process can use a significant amount of disk space. When the disk space is insufficient, exercise caution when performing the ALTER TABLE operation on large tables to prevent the cluster from being read-only.</p> <ul style="list-style-type: none">• Change the data type of a column.• Add columns (including the oid column) to a row-store table.• Modify COLVERSION for a column-store table.• Specify the DEFAULT constant values for a column added to a column-store table. If the DEFAULT values contain volatile functions or are not NULL and do not belong to a specific data type, ensure they are correctly defined. |
| Other | <ul style="list-style-type: none">• When you modify the COLVERSION or enable_delta parameter of a column-store table, other ALTER operations cannot be performed.• You cannot specify a tablespace when running ALTER TABLE for an unlogged table. When you run ALTER TABLE for a non-unlogged table, do not specify pg_unlogged as the tablespace.• Distribution columns cannot be modified for global temporary tables. |

**WARNING**

- Avoid performing **ALTER TABLE**, **ALTER TABLE PARTITION**, **DROP PARTITION**, and **TRUNCATE** operations during peak hours to prevent long SQL statements from blocking these operations or SQL services.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

ALTER TABLE Syntax Format

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
    action [, ... ];
```

action can be the following modification operations:

- Change the data type of an existing column in the table. You can modify the null constraint, default value, and comment information at the same time.

Only the type conversion of the same category (between values, strings, and time) is allowed.

```
ALTER TABLE [ IF EXISTS ] table_name
  MODIFY ( { column_name data_type | [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ] |
    [ CONSTRAINT constraint_name ] NULL | DEFAULT default_expr | COMMENT 'text' } [, ...] );
```

- Rename the table. Changing the name does not affect the stored data. Tables cannot be renamed across schemas. The new table name can contain the schema name of the original table. If **new_table_name** does not specify a schema, the new table and the original table are in the same schema.

```
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO new_table_name;
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO schema.new_table_name;
```

- Rename the specified column in the table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
  RENAME [ COLUMN ] column_name TO new_column_name;
```

- Rename the constraint of the table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
  RENAME CONSTRAINT constraint_name TO new_constraint_name;
```

- Set the schema of the table.

```
ALTER TABLE [ IF EXISTS ] table_name
  SET SCHEMA new_schema;
```

NOTE

- Currently, **SET SCHEMA** can only set schemas to user schemas. It cannot set a schema to a system internal schema.
- The schema setting moves the table into another schema. Associated indexes and constraints owned by table columns are migrated as well. Currently, the schema for sequences cannot be changed. If the table has sequences, delete the sequences, and create them again or delete the ownership between the table and sequences. In this way, the table schema can be changed.
- To change the schema of a table, you must also have CREATE privilege on the new schema. To add the table as a new child of a parent table, you must own the parent table as well. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have CREATE permission on the table's schema. These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the table. However, a system administrator can alter ownership of any table anyway.

- Add columns.

```
ALTER TABLE [ IF EXISTS ] table_name
  ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint
    [ ... ] ] } [, ...] );
```

- Update columns.

```
ALTER TABLE [ IF EXISTS ] table_name
  MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL
    [ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

- Delete an index from a specified table.

```
ALTER TABLE table_name DROP INDEX index_name;
```

 NOTE

- All the actions except for **RENAME** and **SET SCHEMA** can be combined into a list of multiple alterations to apply in parallel. For example, it is possible to add several columns or alter the type of several columns in a single command. This is useful with large tables, since only one pass over the table need be made.
- Adding a **CHECK** or **NOT NULL** constraint requires scanning the table to verify that existing rows meet the constraint.
- Adding a column with a non-null default or changing the type of an existing column will require the entire table to be rewritten. Table rebuilding may take a significant amount of time for a large table; and will temporarily require as much as double the disk space.

Clauses of action Related to Table Operations

There are several clauses of **action**:

```
column_clause
| ADD table_constraint [ NOT VALID ]
| ADD table_constraint_using_index
| VALIDATE CONSTRAINT constraint_name
| DROP CONSTRAINT [ IF EXISTS ] constraint_name [ RESTRICT | CASCADE ]
| CLUSTER ON index_name
| SET WITHOUT CLUSTER
| SET ( {storage_parameter = value} [, ... ] )
| RESET ( storage_parameter [, ... ] )
| OWNER TO new_owner
| SET TABLESPACE new_tablespace
| SET {COMPRESS|NOCOMPRESS}
| DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...] ) } }
| TO { GROUP groupname | NODE ( nodename [, ... ] ) }
| ADD NODE ( nodename [, ... ] )
| DELETE NODE ( nodename [, ... ] )
| DISABLE TRIGGER [ trigger_name | ALL | USER ]
| ENABLE TRIGGER [ trigger_name | ALL | USER ]
| ENABLE REPLICA TRIGGER trigger_name
| ENABLE ALWAYS TRIGGER trigger_name
| DISABLE ROW LEVEL SECURITY
| ENABLE ROW LEVEL SECURITY
| FORCE ROW LEVEL SECURITY
| NO FORCE ROW LEVEL SECURITY
| REFRESH STORAGE
```

- **ADD table_constraint [NOT VALID]**
Adds a new table constraint. When used with the **NOT VALID** option, this constraint is valid only for foreign keys and **CHECK** constraints. If the **NOT VALID** option is added to the constraint, the check on whether the existing records in the table meet the initial constraint is skipped.
- **ADD table_constraint_using_index**
Adds primary key constraint or unique constraint based on the unique index.
- **VALIDATE CONSTRAINT constraint_name**
Validates a foreign key or check constraint that was previously created as **NOT VALID**, by scanning the table to ensure there are no rows for which the constraint is not satisfied. Nothing happens if the constraint is already marked valid.
- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**
Drops a table constraint.

- **CLUSTER ON index_name**
Selects the default index for future **CLUSTER** operations. It does not actually re-cluster the table.
- **SET WITHOUT CLUSTER**
Removes the most recently used **CLUSTER** index specification from the table. This operation affects future cluster operations that do not specify an index.
- **SET ({storage_parameter = value} [, ...])**
Changes one or more storage parameters for the table.
- **RESET (storage_parameter [, ...])**
Resets one or more storage parameters to their defaults. As with **SET**, a table rewrite might be needed to update the table entirely.
- **OWNER TO new_owner**
Changes the owner of the table, sequence, or view to the specified user.
- **SET {COMPRESS|NOCOMPRESS}**
Sets the compression feature of a table. The table compression feature affects only the storage mode of data inserted in a batch subsequently and does not affect storage of existing data. Setting the table compression feature will result in the fact that there are both compressed and uncompressed data in the table.
- **DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH (column_name [,...]) } }**
Changing a table's distribution mode will physically redistribute the table data based on the new distribution mode. After the distribution mode is changed, you are advised to manually run the **ANALYZE** statement to collect new statistics about the table.

 NOTE

- This operation is a major change operation, involving table distribution information modification and physical data redistribution. During the modification, services are blocked. After the modification, the original execution plan of services will change. Perform this operation according to the standard change process.
- This operation is a resource-intensive operation. If you need to modify the distribution mode of large tables, perform the operation when the computing and storage resources are sufficient. Ensure that the remaining space of the entire cluster and the tablespace where the original table is located is sufficient to store a table that has the same size as the original table and is distributed in the new distribution mode.
- **TO { GROUP groupname | NODE (nodename [, ...]) }**
The syntax is only available in extended mode (when GUC parameter **enable_cluster_resize** is **on**). Exercise caution when enabling the mode. It is used for tools like internal dilatation tools. Common users should not use the mode.
- **ADD NODE (nodename [, ...])**
It is only available for tools like internal dilatation. General users should not use the mode.
- **DELETE NODE (nodename [, ...])**
It is only available for internal scale-in tools. Common users should not use the syntax.

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

Disables a single trigger specified by **trigger_name**, disables all triggers, or disables only user triggers (excluding internally generated constraint triggers, for example, deferrable unique constraint triggers and exclusion constraints triggers).

 NOTE

Exercise caution when using this function because data integrity cannot be ensured as expected if the triggers are not executed.

- **ENABLE TRIGGER [trigger_name | ALL | USER]**

Enables a single trigger specified by **trigger_name**, enables all triggers, or enables only user triggers.

- **ENABLE REPLICA TRIGGER trigger_name**

Determines that the trigger firing mechanism is affected by the configuration variable **session_replication_role**. When the replication role is **origin** (default value) or **local**, a simple trigger is fired.

When **ENABLE REPLICA** is configured for a trigger, it is fired only when the session is in **replica** mode.

- **ENABLE ALWAYS TRIGGER trigger_name**

Determines that all triggers are fired regardless of the current replication mode.

- **DISABLE/ENABLE ROW LEVEL SECURITY**

Enables or disables row-level access control for a table.

If row-level access control is enabled for a data table but no row-level access control policy is defined, the row-level access to the data table is not affected. If row-level access control for a table is disabled, the row-level access to the table is not affected even if a row-level access control policy has been defined. For details, see [CREATE ROW LEVEL SECURITY POLICY](#).

 NOTE

Row-level access control cannot be enabled for HDFS tables, foreign tables, and temporary tables.

- **NO FORCE/FORCE ROW LEVEL SECURITY**

Forcibly enables or disables row-level access control for a table.

By default, the table owner is not affected by the row-level access control feature. However, if row-level access control is forcibly enabled, the table owner (excluding system administrators) will be affected. System administrators are not affected by any row-level access control policies.

- **REFRESH STORAGE**

Changes the local hot partitions that meet the criteria specified in the **storage_policy** parameter of an OBS multi-temperature table to the cold partitions stored in the OBS.

For example, if **storage_policy** is set to **'LMT:10'** for an OBS hot or cold table when it is created, the partitions that are not updated within the last 10 days are switched to cold partitions in the OBS.

Clauses of column_clause Related to Column Operations

There are several clauses of **column_clause**:

```
ADD [ COLUMN ] column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint  
[ ... ] ]  
| MODIFY [ COLUMN ] column_name data_type  
| MODIFY [ COLUMN ] column_name [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ]  
| MODIFY [ COLUMN ] column_name [ CONSTRAINT constraint_name ] NULL  
| MODIFY [ COLUMN ] column_name DEFAULT default_expr  
| MODIFY [ COLUMN ] column_name ON UPDATE on_update_expr  
| MODIFY [ COLUMN ] column_name COMMENT comment_text  
| DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]  
| ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING  
expression ]  
| ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }  
| ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL  
| ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer  
| ADD STATISTICS (( column_1_name, column_2_name [, ...] ))  
| ADD { INDEX | UNIQUE [ INDEX ] } [ index_name ] ( ( { column_name | ( expression ) } [ COLLATE  
collation ] [ opclass ] [ ASC | DESC ] [ NULLS LAST ] } [, ...] ) [ USING method ] [ NULLS [ NOT ] DISTINCT  
| NULLS IGNORE ] [ COMMENT 'text' ] LOCAL [ ( { PARTITION index_partition_name } [, ...] ) ] [ WITH  
( { storage_parameter = value } [, ...] ) ]  
| ADD { INDEX | UNIQUE [ INDEX ] } [ index_name ] ( ( { column_name | ( expression ) } [ COLLATE  
collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } } [, ...] ) [ USING method ] [ NULLS [ NOT ]  
DISTINCT | NULLS IGNORE ] [ COMMENT 'text' ] [ WITH ( {storage_parameter = value} [, ...] ) ] [ WHERE  
predicate ]  
| DROP { INDEX | KEY } index_name  
| CHANGE [ COLUMN ] old_column_name new_column_name data_type [ [ CONSTRAINT  
constraint_name ] NOT NULL [ ENABLE ] |  
[ CONSTRAINT constraint_name ] NULL | DEFAULT default_expr | COMMENT 'text' ]  
| DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))  
| ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ...] )  
| ALTER [ COLUMN ] column_name RESET ( attribute_option [, ...] )  
| ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

- **ADD [COLUMN] column_name data_type [compress_mode] [COLLATE collation] [column_constraint [...]]**
Adds a column to a table. If a column is added with **ADD COLUMN**, all existing rows in the table are initialized with the column's default value (**NULL** if no **DEFAULT** clause is specified).
- **ADD ({ column_name data_type [compress_mode] } [, ...])**
Adds columns in the table.
- **MODIFY [COLUMN] column_name data_type**
Modifies the data type of an existing field in a table. Note that the data type of the distribution column cannot be modified.
- **MODIFY [COLUMN] column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE]**
Adds a NOT NULL constraint to a column of a table. Currently, this clause is unavailable to column-store tables.
- **MODIFY [COLUMN] column_name [CONSTRAINT constraint_name] NULL**
Deletes the NOT NULL constraint to a certain column in the table.
- **MODIFY [COLUMN] column_name DEFAULT default_expr**
Changes the default value of the table.
- **MODIFY [COLUMN] column_name ON UPDATE on_update_expr**
Modifies the ON UPDATE expression of a specified column in a table. The column must be of the timestamp or timestampz type. If **on_update_expr** is **NULL**, the **ON UPDATE** clause is deleted.

- **MODIFY [COLUMN] column_name COMMENT comment_text**
Modifies the comment of the table.
- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**
Drops a column from a table. Index and constraint related to the column are automatically dropped. If an object not belonging to the table depends on the column, **CASCADE** must be specified, such as foreign key reference and view.
The **DROP COLUMN** form does not physically remove the column, but simply makes it invisible to SQL operations. Subsequent insert and update operations in the table will store a **NULL** value for the column. Therefore, column deletion takes a short period of time but does not immediately release the table space on the disks, because the space occupied by the deleted column is not reclaimed. The space will be reclaimed when **VACUUM** is executed.
- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**
Change the data type of a field in the table. Only the type conversion of the same category (between values, character strings, and time) is allowed. Indexes and simple table constraints on the column will automatically use the new data type by reparsing the originally supplied expression.
ALTER TYPE requires an entire table be rewritten. This is an advantage sometimes, because it frees up unnecessary space from a table. For example, to reclaim the space occupied by a deleted column, the fastest method is to use the command.

```
ALTER TABLE table ALTER COLUMN anycol TYPE anytype;
```


In this command, **anycol** indicates any column existing in the table and **anytype** indicates the type of the prototype of the column. **ALTER TYPE** does not change the table except that the table is forcibly rewritten. In this way, the data that is no longer used is deleted.
USING in **ALTER TYPE** can specify any expression involving the old values of the row; that is, it can refer to any columns other than the one being converted. This allows very general conversions to be done with the **ALTER TYPE** syntax. Because of this flexibility, the **USING** expression is not applied to the column's default value (if any); the result might not be a constant expression as required for a default. This means that when there is no implicit or assignment cast from old to new type, **ALTER TYPE** might fail to convert the default even though a **USING** clause is supplied. In such cases, drop the default with **DROP DEFAULT**, perform the **ALTER TYPE**, and then use **SET DEFAULT** to add a suitable new default. Similar considerations apply to indexes and constraints involving the column.
- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**
Sets or removes the default value for a column. The default values only apply to subsequent **INSERT** commands; they do not cause rows already in the table to change. Defaults can also be created for views, in which case they are inserted into **INSERT** statements on the view before the view's **ON INSERT** rule is applied.
- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**
Changes whether a column is marked to allow **NULL** values or to reject **NULL** values. You can only use **SET NOT NULL** when the column contains no **NULL** values.

- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**
Specifies the per-column statistics-gathering target for subsequent **ANALYZE** operations. The value ranges from **0** to **10000**. Set it to **-1** to revert to using the default system statistics target.
- **ADD { INDEX | UNIQUE [INDEX] } [index_name] ({ { column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS LAST] } [, ...]) [USING method] [NULLS [NOT] DISTINCT | NULLS IGNORE] [COMMENT 'text'] LOCAL [({ PARTITION index_partition_name } [, ...])] [WITH ({ storage_parameter = value } [, ...])]**
Create an index for the partitioned table. For details about the parameters, see [CREATE INDEX](#).
- **ADD { INDEX | UNIQUE [INDEX] } [index_name] ({ { column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS { FIRST | LAST }] } [, ...]) [USING method] [NULLS [NOT] DISTINCT | NULLS IGNORE] [COMMENT 'text'] [WITH ({storage_parameter = value} [, ...])] [WHERE predicate]**
Create an index on the table. For details about the parameters, see [CREATE INDEX](#).
- **DROP { INDEX | KEY } index_name**
Deletes an index from a table.
- **CHANGE [COLUMN] old_column_name new_column_name data_type [[CONSTRAINT constraint_name] NOT NULL [ENABLE] | [CONSTRAINT constraint_name] NULL | DEFAULT default_expr | COMMENT 'text']**
Modifies the column information in the table, such as column names and column field information.
- **{ADD | DELETE} STATISTICS ((column_1_name, column_2_name [, ...]))**
Adds or deletes the declaration of collecting multi-column statistics to collect multi-column statistics as needed when **ANALYZE** is performed for a table or a database. The statistics about a maximum of 32 columns can be collected at a time. You are not allowed to add or delete the declaration for system tables or foreign tables
- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**
ALTER [COLUMN] column_name RESET (attribute_option [, ...])
Sets or resets per-attribute options.
The attribute option parameters are **n_distinct**, **n_distinct_inherited**, and **cstore_cu_sample_ratio**. **n_distinct** specifies and fixes the statistics of a table's distinct values. **n_distinct_inherited** specifies and inherits the distinct value statistics. **cstore_cu_sample_ratio** specifies the CU ratio for **ANALYZE** on a column-store table. Currently, the **n_distinct_inherited** parameter cannot be set or reset.
 - **n_distinct**
Sets the distinct value statistics of the column.
Value range: -1.0 to INT_MAX
Default value: **0**, indicating that this parameter is not set.

- **n_distinct_inherited**
Sets the distinct value statistics of the column in an inherited table.
Value range: -1.0 to INT_MAX
Default value: **0**, indicating that this parameter is not set.
- **cstore_cu_sample_ratio**
Specifies the expansion multiple in the calculation of CUs to be sampled during ANALYZE on a column-store table.
Value range: 1.0-10000.0
Default value: **1.0**
- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**
Sets the storage mode for a column. This clause specifies whether this column is held inline or in a secondary TOAST table, and whether the data should be compressed. This statement can only be used for row-based tables. **SET STORAGE** only sets the strategy to be used for future table operations.

Options of Other Parameters

- **column_constraint** is as follows:
[CONSTRAINT constraint_name]
{ NOT NULL |
NULL |
CHECK (expression) |
DEFAULT default_expr |
UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] index_parameters |
PRIMARY KEY index_parameters }
[DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE]
- **compress_mode** of a column is as follows:
[DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS]
- **table_constraint_using_index** used to add the primary key constraint or unique constraint based on the unique index is as follows:
[CONSTRAINT constraint_name]
{ UNIQUE | PRIMARY KEY } USING INDEX index_name
[DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE]
- Add foreign key constraint **REFERENCES**.
[CONSTRAINT constraint_name]
FOREIGN KEY (column_name [, ...]) REFERENCES reftable [(refcolumn [, ...])]
- **table_constraint** is as follows:
[CONSTRAINT constraint_name]
{ CHECK (expression) |
UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] (column_name [, ...]) index_parameters |
PRIMARY KEY (column_name [, ...]) index_parameters }
[DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE]
- **index_parameters** is as follows:
[WITH ({storage_parameter = value} [, ...])]
[USING INDEX TABLESPACE tablespace_name]

Parameter Description

Table 21-159 ALTER TABLE parameters

| Parameter | Description | Value Range |
|---|--|---|
| IF EXISTS | Sends a prompt instead of an error if the table name does not exist. | - |
| ONLY | Performs operations only on the specified table name. If ONLY is not specified:

The table and all subtables will be modified.

Note that you can only add or delete columns to or from a parent or child table, or change the type of a column. The parent table and its subsequent representatives must always have the same columns and types. | - |
| table_name | Specifies the name of the table to be modified.

You can add the asterisk (*) option following the table name to specify that all subtables are scanned, which is the default operation. | Valid table name. |
| new_table_name | New name of a table. | A string compliant with the identifier naming rules . |
| column_name, column_1_name, and column_2_name | Specifies the name of an existing or new field. | Valid column name. |
| new_column_name | Specifies the new name of a column. | A string compliant with the identifier naming rules . |
| constraint_name | Specifies the name of a constraint. | Valid constraint name. |
| new_constraint_name | Specifies the new name of a table constraint. | A string compliant with the identifier naming rules . |
| schema_name | Specifies the schema name of a table. | Valid schema name. |
| new_schema | Specifies the new schema name. | Valid schema name. |
| index_name | Specifies the index name. | Valid index name. |

| Parameter | Description | Value Range |
|-------------------|---|---|
| new_owner | Specifies the name of the new owner of the table. | Valid username or role name. |
| new_tablespace | Specifies the name of the new tablespace to which the table belongs. | Valid tablespace name. |
| data_type | Specifies the type of a new column or a new type of an existing column. | A string compliant with the identifier naming rules . |
| storage_parameter | Specifies the storage parameters of a table. | For details, see Table 21-160 . |
| compress_mode | Specifies the compression option of the table column. Currently, this parameter is valid only for row-store tables. The clause specifies the algorithm preferentially used by the column. | - |
| collation | Specifies the collation rule name of a column. The optional COLLATE clause specifies a collation for the new column; if omitted, the collation is the default for the new column. | - |
| USING expression | A USING clause specifies how to compute the new column value from the old; if omitted, the default conversion is an assignment cast from old data type to new. A USING clause must be provided if there is no implicit or assignment cast from the old to new type. | - |
| NOT NULL NULL | Sets whether the column allows null values. | - |
| integer | Specifies the constant value of an integer with a sign. When PERCENT is used, statistics are collected based on the percentage of table data. | 0~100 |
| attribute_option | Specifies an attribute option. | - |

| Parameter | Description | Value Range |
|---------------------------------------|---|-------------|
| PLAIN EXTERNAL
EXTENDED MAIN | <p>Specifies a column storage mode.</p> <ul style="list-style-type: none">• PLAIN must be used for fixed-length values (such as integers). It must be inline and uncompressed.• MAIN is for inline, compressible data.• EXTERNAL is for external, uncompressed data. Use of EXTERNAL will make substring operations on text and bytea values run faster, at the penalty of increased storage space.• EXTENDED is for external, compressed data. EXTENDED is the default for most data types that support non-PLAIN storage. | - |
| CHECK
(expression) | <p>New or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to TRUE succeed. If any row of an insert or update operation produces a FALSE result, an error exception is raised and the insert or update does not alter the database.</p> <p>A check constraint specified as a column constraint should reference only the column's values, while an expression appearing in a table constraint can reference multiple columns.</p> <p>Currently, CHECK expression does not include subqueries and cannot use variables apart from the current column.</p> | - |

| Parameter | Description | Value Range |
|---|--|-------------|
| DEFAULT
default_expr | <p>Assigns a default data value for a column.</p> <p>The data type of the default expression must match the data type of the column.</p> <p>The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is NULL.</p> <p>If a suffix operator, such as (!). is used in default_expr, enclose the operator in parentheses.</p> | - |
| UNIQUE [NULLS
[NOT] DISTINCT
NULLS IGNORE]
index_parameters

UNIQUE
(column_name
[, ...]) [NULLS
[NOT] DISTINCT
NULLS IGNORE]
index_parameters | <p>The UNIQUE constraint specifies that a group of one or more columns of a table can contain only unique values.</p> <p>The [NULLS [NOT] DISTINCT NULLS IGNORE] field is used to specify how to process null values in the index column of the Unique index. For details, see Table 21-161.</p> <p>Default value: This parameter is left empty by default. NULL values can be inserted repeatedly.</p> | - |
| PRIMARY KEY
index_parameters

PRIMARY KEY
(column_name
[, ...])
index_parameters | <p>The primary key constraint specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.</p> | - |

| Parameter | Description | Value Range |
|--|---|-------------|
| DEFERRABLE NOT DEFERRABLE INITIALLY DEFERRED INITIALLY IMMEDIATE | Sets whether the constraint is deferrable. This option is unavailable to column-store tables. <ul style="list-style-type: none">• DEFERRABLE: deferrable can be postponed until the end of the transaction using the SET CONSTRAINTS command.• NOT DEFERRABLE: checks immediately after the execution of each command.• INITIALLY IMMEDIATE: checks immediately after the execution of each statement.• INITIALLY DEFERRED: checks when the transaction ends. | - |
| WITH
({storage_parameter = value} [, ...]) | Specifies an optional storage parameter for a table or an index. | - |
| COMPRESS
NOCOMPRESS | <ul style="list-style-type: none">• NOCOMPRESS: If the NOCOMPRESS keyword is specified, the existing compression feature of the table is not changed.• COMPRESS: If the COMPRESS keyword is specified, the table compression feature is triggered if tuples are inserted in a batch. | - |
| CASCADE | Automatically drops objects that depend on the dropped column or constraint (for example, views referencing the column). | - |
| RESTRICT | Refuses to drop the column or constraint if there are any dependent objects. This is the default behavior. | - |

| Parameter | Description | Value Range |
|--------------|---|---|
| cache_policy | <p>Specifies the table cache policy. This parameter is supported only in storage-compute decoupled 3.0 clusters.</p> <p>Default value: ALL</p> | <ul style="list-style-type: none">• ALL: Hot cache is used for the entire table.• NONE: Cold cache is used for the entire table.• HPN: The first <i>N</i> partitions in a partitioned table use hot cache. The rest of the partitions use cold cache.• HPL: <i>P1, P2,</i> In a partitioned table, the specified partitions use hot cache. The rest of the partitions use cold cache. <p>NOTE</p> <ul style="list-style-type: none">• For foreign tables and non-partitioned tables, only the ALL and NONE cache policies are supported.• Only range-partitioned and list-partitioned internal tables support HPN and HPL cache policies. |

Table 21-160 storage_parameter parameters

| Parameter Type | Parameter | Description | Value Range |
|---|-----------|---|---|
| Partition management parameters (newly added) | PERIOD | <p>Sets the period for automatically creating partitions in partition management.</p> <p>NOTE</p> <ul style="list-style-type: none">If this parameter is not configured when you create a table, you can run the set statements to configure this parameter and enable automatic partition creation. If this parameter has been configured before, you can run the set statements to modify this parameter.You can run the reset command to disable the automatic partition creation function. However, if the automatic partition deletion function is enabled, the automatic partition creation function cannot be disabled. | <p>Interval type.</p> <p>For details about the value range of PERIOD and the constraints on enabling this function, see the PERIOD parameter in WITH Clause Parameters.</p> |
| | TTL | <p>Sets the partition expiration time for automatically deleting partitions in partition management.</p> <p>NOTE</p> <ul style="list-style-type: none">If this parameter is not configured when you create a table, you can run the set statements to configure this parameter and enable automatic partition deletion. If this parameter has been configured before, you can run the set statements to modify this parameter.You can run the reset command to disable the automatic partition deletion. | <p>Interval type.</p> <p>For details about the value range of TTL and restrictions on enabling this function, see the TTL parameters in WITH Clause Parameters.</p> |

| Parameter Type | Parameter | Description | Value Range |
|---|--------------------|---|-------------|
| Parameters related to column-store tables in Turbo storage format (newly added) | enable_turbo_store | <p>Specifies whether the column-store table is in the turbo storage format. This is supported only by 9.1.0.100 and later cluster versions.</p> <p>NOTE</p> <ul style="list-style-type: none">Common column-store tables in version 3.0 do not support the turbo storage format, while HStore Opt tables in version 3.0 only support the turbo storage format.In version 2.0, there is no restriction on column-store tables. | - |

The NULL value of the index column in the unique index is processed as follows:

Use the **[NULLS [NOT] DISTINCT | NULLS IGNORE]** field to set how null values are handled in the unique index column.

When the inserted data is compared with the original data in the table, the NULL value can be processed in any of the following ways:

- **NULLS DISTINCT:** NULL values are unequal and can be inserted repeatedly.
- **NULLS NOT DISTINCT:** NULL values are equal. If all index columns are NULL, NULL values cannot be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different.
- **NULLS IGNORE:** NULL values are skipped during the equivalent comparison. If all index columns are NULL, NULL values can be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different.

Table 21-161 Processing of NULL values in index columns in unique indexes

| Constraint | All Index Columns Are NULL | Some Index Columns Are NULL. |
|--------------------|--------------------------------|--|
| NULLS DISTINCT | Can be inserted repeatedly. | Can be inserted repeatedly. |
| NULLS NOT DISTINCT | Cannot be inserted repeatedly. | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |

| Constraint | All Index Columns Are NULL | Some Index Columns Are NULL. |
|--------------|-----------------------------|--|
| NULLS IGNORE | Can be inserted repeatedly. | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |

Table Operation Examples

Create the sample table **customer**.

```
DROP TABLE IF EXISTS customer;
CREATE TABLE customer
(
  C_CUSTKEY    BIGINT      ,
  C_NAME      VARCHAR(25) ,
  C_ADDRESS   VARCHAR(40) ,
  C_NATIONKEY INT          ,
  C_PHONE     CHAR(15)    ,
  C_ACCTBAL   DECIMAL(15,2)
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Create the column-store table **customer_address**.

```
DROP TABLE IF EXISTS customer_address;
CREATE TABLE 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)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH,COLVERSION=2.0)
DISTRIBUTE BY HASH (ca_address_sk);
```

- Change the schema of a table.
Set the schema of the **customer** table to **myschema**.

```
ALTER TABLE customer SET SCHEMA myschema;
```
- Add a primary key constraint or unique constraint based on the unique index.
Create a unique index **customer_constraint1** for the customer table, add a primary key constraint based on the existing unique index, and rename the index.

```
CREATE UNIQUE INDEX customer_constraint1 ON customer(C_CUSTKEY);
ALTER TABLE customer ADD CONSTRAINT customer_constraint2 PRIMARY KEY USING INDEX customer_constraint1;
```
- Rename a table constraint.

```
ALTER TABLE customer RENAME CONSTRAINT customer_constraint2 TO customer_constraint;
```
- Delete a table constraint.

```
ALTER TABLE customer DROP CONSTRAINT customer_constraint;
```
- Add a primary key constraint to a table.

```
ALTER TABLE CUSTOMER ADD PRIMARY KEY(C_CUSTKEY);
```
- Add a partial cluster column to the column-store table **customer_address**.

```
ALTER TABLE customer_address ADD CONSTRAINT customer_address_cluster PARTIAL CLUSTER KEY(ca_address_sk);
```

- Delete the partial cluster column from the column-store table **customer_address**.
`ALTER TABLE customer_address DROP CONSTRAINT customer_address_cluster;`
- Modify the storage format of a column-store table.
`ALTER TABLE customer_address SET (COLVERSION = 1.0);`
- Change the distribution mode of a table.
`ALTER TABLE customer_address DISTRIBUTE BY REPLICATION;`
- Rename a table.
`ALTER TABLE CUSTOMER RENAME TO CUSTOMER_t;`
- Modify the table cache policy (supported only in storage-compute decoupled 3.0 clusters).
`ALTER TABLE orders SET (cache_policy = 'NONE');`
- Disable the automatic partition deletion function of a table. For details about how to create a table, see [Example: Creating a Partitioned Table with Automatic Partition Management and Specified Partitions](#).
`ALTER TABLE CPU RESET (ttl);`

Index Operation Example

- Add an index to a table.
`ALTER TABLE customer ADD INDEX customer_index(C_CUSTKEY);`
- Delete a table index.
`ALTER TABLE customer DROP INDEX customer_index;`
`ALTER TABLE customer DROP KEY customer_index;`
- Adds an index to a column in a table.
`ALTER TABLE CUSTOMER ADD c_address_id varchar(20) CONSTRAINT ca_address_index CHECK (c_address_id > 0);`

Hot and Cold Table Operation Example

- Do a cold or hot switchover for a single table.
`ALTER TABLE cold_hot_table REFRESH STORAGE;`
- Change a column-store partitioned table to a hot and cold table.
`CREATE table test_1(id int,d_time date)
WITH(ORIENTATION=COLUMN)
DISTRIBUTE BY HASH (id)
PARTITION BY RANGE (d_time)
(PARTITION p1 START('2022-01-01') END('2022-01-31') EVERY(interval '1 day'))

ALTER TABLE test_1 SET (storage_policy = 'LMT:100');`

Column Operation Example

Create the sample table **warehouse_t**.

```
DROP TABLE IF EXISTS warehouse_t;  
CREATE TABLE warehouse_t  
(  
    W_WAREHOUSE_SK      INTEGER          NOT NULL,  
    W_WAREHOUSE_ID      CHAR(16)         NOT NULL,  
    W_WAREHOUSE_NAME     VARCHAR(20)     UNIQUE DEFERRABLE,  
    W_WAREHOUSE_SQ_FT    INTEGER          ,  
    W_COUNTY            VARCHAR(30)      ,  
    W_STATE              CHAR(2)         DEFAULT 'GA',  
    W_ZIP               CHAR(10)  
);
```

- Adds a column.
Adds a column to the **warehouse_t** table.
`ALTER TABLE warehouse_t ADD W_GOODS_CATEGORY int;`

Adds multiple columns to the **warehouse_t** table.

```
ALTER TABLE warehouse_t ADD (W_COMMENT VARCHAR(117) NOT NULL, W_COUNT int);
```

- Delete a column.

Delete a column from the **warehouse_t** table.

```
ALTER TABLE warehouse_t DROP COLUMN W_STATE;
```

- Rename a column.

Change the column name **W_ZIP** of the **warehouse_t** table to **new_W_ZIP**.

```
ALTER TABLE warehouse_t RENAME W_ZIP TO new_W_ZIP;
```

- Change the column name and column field information.

Change the column name **W_GOODS_CATEGORY** of the **warehouse_t** table to **W_GOODS_CATEGORY2**, change the data type of the **W_GOODS_CATEGORY2** column to **DECIMAL**, and set the column constraint to **NOT NULL**.

```
ALTER TABLE warehouse_t CHANGE W_GOODS_CATEGORY W_GOODS_CATEGORY2 DECIMAL NOT NULL COMMENT 'W_GOODS_CATEGORY';
```

- Change the comment of a column.

```
ALTER TABLE warehouse_t MODIFY W_GOODS_CATEGORY2 COMMENT 'W_GOODS_CATEGORY';
```

- Change the data type of a column.

Change the data type of the **warehouse_t** column and set the column constraint to **NOT NULL**.

```
ALTER TABLE warehouse_t MODIFY W_WAREHOUSE_SQ_FT varchar(20) NOT NULL;
```

- Add the **NOT NULL** constraint to a column.

```
ALTER TABLE warehouse_t ALTER COLUMN W_COUNTY SET NOT NULL;
```

- Add a primary key to the **warehouse_t** table.

```
ALTER TABLE warehouse_t ADD PRIMARY KEY(w_warehouse_name);
```

- Modify the **ON UPDATE** expression of a specified column in a table.

Add a timestamp column with the **ON UPDATE** expression to the **warehouse_t** table. The timestamp of the column is automatically updated when an **UPDATE** operation is performed on the table.

```
ALTER TABLE warehouse_t ADD COLUMN C_TIME timestamp on update current_timestamp;
```

Delete the timestamp column with the **ON UPDATE** expression from the **warehouse_t** table.

```
ALTER TABLE warehouse_t MODIFY COLUMN C_TIME timestamp on update NULL;
```

Helpful Links

[CREATE TABLE](#), [12.101-RENAME TABLE](#), and [DROP TABLE](#)

21.13.4.5 ALTER TABLE PARTITION

Function

ALTER TABLE PARTITION is used to modify table partitions, including adding, deleting, splitting, merging partitions, migrating data from ordinary tables to partitioned tables, and modifying partition attributes.

A partitioned table is a logical table that is divided into segments, called partitions whose data is stored on different physical blocks.

Common partitioning strategies define data ranges based on one or more columns. Each partition stores data within a range. These columns are called partition keys.

Currently, DWS row-store and column-store tables support range partitioning and list partitioning.

- Range partitioning

A table is partitioned into ranges defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.

 - Core feature: Data is distributed by consecutive intervals. It is used in scenarios where data is ordered and retrieved by range.
 - Scenario: Partition keys increase linearly or have continuous intervals. Data query is usually performed around a certain range, and new data naturally falls into a new range. For example, data about orders is partitioned by time range, and data about user levels is partitioned by value range.
 - Range partitioning maps data to partitions based on ranges of values of the partitioning key that you establish for each partition. This is the most commonly used partitioning policy. Currently, range partitioning only allows the use of the range partitioning policy.
- List partitioning

List partitioning allocates records to partitions based on the key values in each partition. The key values do not overlap in different partitions. Each set of key values gets its own partition for storing related data. List partitioning is supported only by clusters of version 8.1.3 or later.

 - Core feature: Data is divided based on discrete enumerated values, which are used as fixed classification category or high-frequency filter criteria.
 - Scenario: Partition key values are fixed and discrete. Data query is usually based on a certain category, and key values do not increase irregularly. For example, the user information table is partitioned by province, and each province corresponds to a partition.
 - In list partitioning, data is mapped to a created partition based on the partition key value. If the data can be mapped to, it is inserted into the specific partition. If it cannot be mapped to, error messages are returned.

Precautions

- The name of the partition to be added cannot be the same as that of an existing partition in the partitioned table.
- For a range partitioned table, the boundary value of the added partition must be the same type as the partition key of the partitioned table. The key value of the added partition must exceed the upper limit of the last partition.
- For a list partitioned table, if the DEFAULT partition has been defined, no new partition can be added.
- Unless otherwise specified, the syntax of range partitioned tables is the same as that of column-store partitioned tables.
- If the number of partitions in the target partitioned table has reached the maximum (32767), partitions cannot be added.
- If a partitioned table has only one partition, the partition cannot be deleted.
- When you run the **DROP PARTITION** command to delete a partition, the data in the partition is also deleted.

- Use **PARTITION FOR()** to choose partitions. The number of specified values in the brackets should be the same as the column number in customized partition, and they must be consistent.
- The **Value** partitioned table does not support the **Alter Partition** operation.
- For OBS hot and cold tables:
 - They do not support specifying the partition table's tablespace as the OBS tablespace for **MOVE**, **EXCHANGE**, **MERGE**, and **SPLIT** operations.
 - When an **ALTER** statement is executed, the data in the cold partition should stay in the cold partition, and the data in the hot partition should remain in the hot partition. It is not allowed to move cold partition data to the local tablespace.
 - Only the default tablespace is supported for cold partitions.
 - Cold and hot partitions cannot be merged.
 - Cold partition switching is not supported during the **EXCHANGE** operation.

**WARNING**

- Avoid performing **ALTER TABLE**, **ALTER TABLE PARTITION**, **DROP PARTITION**, and **TRUNCATE** operations during peak hours to prevent long SQL statements from blocking these operations or SQL services.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

Syntax

Modify the syntax of the table partition.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
action [, ... ];
```

action indicates the following clauses for maintaining partitions. For the partition continuity when multiple clauses are used for partition maintenance, DWS does **DROP PARTITION** and then **ADD PARTITION**, and finally runs the rest clauses in sequence.

- **MODIFY PARTITION: Setting Whether a Partitioned Index Is Available**
- **REBUILD PARTITION syntax: Rebuilding a Partitioned Index**
- **EXCHANGE PARTITION: Moving the Data from an Ordinary Table to a Specified Partition**
- **ROW MOVEMENT: Setting the Row Movement Switch of a Partitioned Table**
- **MERGE PARTITIONS: Merging Partitions Into One**
- **SPLIT PARTITION: Splitting One Partition into Multiple Ones**
- **ADD PARTITION Sub-syntax: Adding One or More Partitions to a Specified Partitioned Table**
- **DROP PARTITION Sub-syntax: Deleting a Specified Partition or Multiple Partitions from a Partitioned Table**

- **TRUNCATE PARTITION: Clearing Data in a Table Partition**

ALTER TABLE PARTITION Main Clause Parameters

Table 21-162 ALTER TABLE PARTITION main clause parameters

| Parameter | Description | Value Range |
|-----------------|--|--|
| table_name | Specifies the name of the partitioned table to be modified. | Name of an existing partitioned table. |
| action [, ...] | Specifies clauses for maintaining partitions. For the partition continuity when multiple clauses are used for partition maintenance, DWS does DROP PARTITION and then ADD PARTITION , and finally runs the rest clauses in sequence. | For details, see the following clauses. |
| partition_name | Specifies the name of the partition to be modified. | Name of an existing partition. |
| partition_value | Specifies the key value of a partition.
The value specified by PARTITION FOR (partition_value [, ...]) can uniquely identify a partition. | Value range of the partition key of the partition to be renamed. |

MODIFY PARTITION: Setting Whether a Partitioned Index Is Available

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
MODIFY PARTITION partition_name { UNUSABLE LOCAL INDEXES | REBUILD UNUSABLE LOCAL INDEXES }
```

Table 21-163 modify_clause parameters

| Parameter | Description | Value Range |
|--------------------------------|---|--------------------------------|
| partition_name | Specifies the name of the partition to be modified. | Name of an existing partition. |
| UNUSABLE LOCAL INDEXES | Sets all the indexes unusable in the partition. | - |
| REBUILD UNUSABLE LOCAL INDEXES | Rebuilds all the indexes in the partition. | - |

REBUILD PARTITION syntax: Rebuilding a Partitioned Index

REBUILD PARTITION can rebuild indexes in a partitioned table. It is mainly used to restore the unusable partitioned indexes caused by abnormal operations (such

as a large number of partition data changes and index damage) or optimize the partitioned index performance. Compared with rebuilding indexes in an entire table, rebuilding indexes in a partitioned table takes less resources and shorter time.

Typical application scenarios

- **Repairing invalid partitioned indexes:** After a partition executes a large number of DML operations (such as batch deletion and update) or is interrupted unexpectedly, the partitioned indexes may become unusable. In this case, you need to rebuild the indexes.
- **Performance optimization:** For a partitioned table that has been running for a long time, indexes may have fragments. Rebuilding the indexes can optimize the index structure and improve the query efficiency.
- **Partition data exchange:** After partition data is exchanged using EXCHANGE PARTITION, indexes may need to be rebuilt to ensure consistency.

This syntax is supported only by clusters of version 8.3.0.100 or later.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
REBUILD PARTITION partition_name [ WITHOUT UNUSABLE ]
```

Table 21-164 rebuild_clause parameters

| Parameter | Description | Value Range |
|------------------|---|-------------|
| WITHOUT UNUSABLE | Ignores indexes in the UNUSABLE state when indexes on a partition are rebuilt. | - |

EXCHANGE PARTITION: Moving the Data from an Ordinary Table to a Specified Partition

EXCHANGE PARTITION is a key operation for efficiently managing partitioned tables in DWS. It can exchange metadata between a partition in a partitioned table and an ordinary table, instead of physically moving data. This means that the ownership of data blocks can be quickly switched, avoiding large-scale I/O operations and greatly improving performance.

Application scenarios:

1. **Batch data loading:** It can load pre-processed data to a temporary table and then replace the data in the target partition, reducing the write pressure on the production table during peak hours.
2. **Historical data archiving:** It can exchange partitions that are not frequently accessed to independent tables, facilitating migration to low-cost storage.
3. **Partition data repair:** It can exchange the data of a faulty partition to a temporary table for correction and then exchange the data back to the partitioned table, reducing lock contention during the repair.
4. **Data migration:** It can exchange the data of a specific partition across tables or databases to avoid inserting data one by one.

Syntax:

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
EXCHANGE PARTITION { ( partition_name ) | FOR ( partition_value [, ...] ) }  
    WITH TABLE {[ ONLY ] ordinary_table_name | ordinary_table_name * | ONLY ( ordinary_table_name )}  
    [ { WITH | WITHOUT } VALIDATION ] [ VERBOSE ]
```

Table 21-165 exchange_clause parameters

| Parameter | Description | Value Range |
|----------------------------------|--|--|
| partition_value | Specifies the key value of a partition.
The value specified by PARTITION FOR (partition_value [, ...]) can uniquely identify a partition. | Value range of the partition key of the partition to be renamed. |
| ordinary_table_name | Specifies the name of the ordinary table whose data is to be migrated. | Name of an existing common table. |
| { WITH WITHOUT }
VALIDATION | Checks whether the ordinary table data meets the specified partition key range of the partition to be exchanged. The default value is WITH .
The check is time consuming, especially when the data volume is large. Therefore, use WITHOUT when you are sure that the current ordinary table data meets the partition key range of the partition to be exchanged. | <ul style="list-style-type: none">● WITH: checks whether the ordinary table data meets the partition key range of the partition to be exchanged. If any data does not meet the required range, an error is reported.● WITHOUT: does not check whether the ordinary table data meets the partition key range of the partition to be exchanged. |
| VERBOSE | When VALIDATION is WITH , if the ordinary table contains data that is out of the partition key range, insert the data to the correct partition. If there is no correct partition where the data can be route to, an error is reported.
CAUTION
Only when VALIDATION is WITH , VERBOSE can be specified. | - |

The ordinary table and the partitioned table whose data is to be exchanged must meet the following requirements:

- The number of columns of the ordinary table is the same as that of the partitioned table, and their information should be consistent, including the column name, data type, constraint, collation, storage parameter, compression, and data type of a deleted column.
- The compression information of the ordinary table and partitioned table should be consistent.
- The distribution column information of the ordinary table and the partitioned table should be consistent.
- The number and information of indexes of the ordinary table and the partitioned table should be consistent.
- The number and information of constraints of the ordinary table and the partitioned table should be consistent.
- The ordinary table cannot be a temporary table or unlogged table.
- Exchanging partitions between an ordinary table and a partitioned table requires them to be in the same logical cluster or node group to avoid slow data insertion. Otherwise, the exchange operation becomes a slow data copy process, especially with large tables.
- In online scale-out and redistribution scenarios, the exchange partition statement may interfere with the redistribution of ordinary tables and partitioned tables (if there are lock conflicts between the partition exchange and redistribution statement). Usually, the redistribution of ordinary tables and partitioned tables is retried twice after being interrupted, but if a table is exchanged too often, the redistribution may fail for multiple times. If the redistribution process of an ordinary table is interrupted by the partition exchange operation, the data has been replaced with the data in the original partitioned table during the redistribution retry. In this case, full redistribution will be performed again.
- If other columns following the last valid column in the partitioned table are deleted and the deleted columns are not considered, the partitioned table can be exchanged with the ordinary table as long as the columns of the two tables are the same.
- The table-level parameter **colversion** must be consistent between the column-store ordinary table and the column-store partitioned table. The colversion 2.0 and colversion 1.0 cannot be exchanged.

When the exchange is done, the data and tablespace of the ordinary table and partitioned table are exchanged. In this case, statistics about the ordinary table and the partitioned table become unreliable. Both tables should be analyzed again.

ROW MOVEMENT: Setting the Row Movement Switch of a Partitioned Table

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
{ ENABLE | DISABLE } ROW MOVEMENT
```

Table 21-166 row_clause parameters

| Parameter | Description | Value Range |
|-----------------------------|---|---|
| ENABLE/DISABLE ROW MOVEMENT | <p>Specifies whether to relocate a row within a table. If the new value of a partition key in a row belongs to another partition, an error message is displayed or the row is moved to the corresponding partition.</p> <ul style="list-style-type: none">• ENABLE: The row is moved to the corresponding partition. If partition keys are frequently updated, enable this parameter.• DISABLE: The row is not moved to the corresponding partition. If partition keys are updated, an error is reported. If data cross-partition update is strictly prohibited, disable this parameter. <p>The default value is DISABLE.</p> <p>CAUTION
To enable cross-partition update, you need to enable row movement. However, if SELECT FOR UPDATE is executed concurrently to query the partitioned table, the query results may be inconsistent. Therefore, exercise caution when performing this operation.</p> | <p>The date column (partition key) of a partitioned table is partitioned by quarter into p_2023q1 and p_2023q2. The value 2023-02-15 of a row belongs to the partition p_2023q1 in the first quarter. After the value is updated to 2023-05-15, you can configure ROW MOVEMENT to determine whether to move the data in this row.</p> <ul style="list-style-type: none">• ENABLE: The row movement is enabled. The data is moved to the new partition p_2023q2.• DISABLE: The row movement is disabled. The system reports an error. <p>For details, see Examples: Enabling and Disabling ROW MOVEMENT.</p> |

MERGE PARTITIONS: Merging Partitions Into One

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
MERGE PARTITIONS { partition_name } [, ...] INTO PARTITION partition_name
```

- The partition before the keyword **INTO** is called the source partition, and the partition after the **INTO** is called the target partition.
- The number of source partitions cannot be less than 2.
- The source partition name must be unique.
- The source partition cannot have unusable indexes. Otherwise, an error will be reported.
- The target partition name must either be the same as the name of the last source partition or different from all partition names of the table.

- The boundaries of the target partition are the union of the boundaries of all the source partitions.
- For a range partitioned table, all source partitions must have contiguous boundaries.
- For list partitioning, if the source partition contains a DEFAULT partition, the boundary of the target partition is also DEFAULT.

SPLIT PARTITION: Splitting One Partition into Multiple Ones

split_clause specifies range partitioning.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_point_clause |
no_split_point_clause }
```

- **split_point_clause:**

```
AT ( partition_value ) INTO ( PARTITION partition_name , PARTITION partition_name )
```

CAUTION

The size of split point should be in the range of splitting partition key. The split point can only split one partition into two.

- **no_split_point_clause:**

```
INTO { ( partition_less_than_item [, ...] ) | ( partition_start_end_item [, ...] ) }
```

CAUTION

- The first new partition key specified by **partition_less_than_item** must be larger than that of the former partition (if any), and the last partition key specified by **partition_less_than_item** must be equal to that of the splitting partition.
 - The start point (if any) of the first new partition specified by **partition_start_end_item** must be equal to the partition key (if any) of the previous partition. The end point (if any) of the last partition specified by **partition_start_end_item** must be equal to the partition key of the splitting partition.
 - **partition_less_than_item** supports a maximum of four partition keys and **partition_start_end_item** supports only one partition key. For details about the supported data types, see the **PARTITION BY RANGE(partition_key)** parameter in [Table 21-150](#).
 - **partition_less_than_item** and **partition_start_end_item** cannot be used in the same statement. There is no restriction on different SPLIT statements.
- **partition_less_than_item:**
PARTITION partition_name VALUES LESS THAN ({ partition_value | MAXVALUE } [, ...])
 - **partition_start_end_item** (For details about the restrictions, see the **partition_start_end_item** parameter in [Table 21-150](#)).
PARTITION partition_name {
 {START(partition_value) END (partition_value) EVERY (interval_value)} |


```
{START(partition_value) END ({partition_value | MAXVALUE})} |  
{START(partition_value)} |  
{END({partition_value | MAXVALUE})}  
}
```

split_clause specifies list partitioning.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
SPLIT PARTITION { partition_name | FOR ( partition_value [, ...] ) } { split_values_clause |  
split_no_values_clause }
```

- **split_values_clause** specifies a split point.

```
VALUES ( { (partition_value) [, ...] } | DEFAULT ) INTO ( PARTITION partition_name , PARTITION  
partition_name )
```

CAUTION

- If the source partition is not a **DEFAULT partition**, the boundary specified by the cut point is a non-null true subset of the source partition boundary. If the source partition is a **DEFAULT** partition, the boundary specified by the cut point cannot overlap with the boundaries of other non-**DEFAULT** partitions.
- The boundary specified by the split point is the boundary of the first partition after the keyword **INTO**. The difference between the boundary of the source partition and the specified boundary of the split point is the boundary of the second partition.
- If the source partition is the **DEFAULT** partition, the boundary of the second partition is still **DEFAULT**.

-
- **split_no_values_clause** specifies no split point is specified.

```
INTO ( list_partition_item [, ...], PARTITION partition_name )
```

CAUTION

- The syntax of **list_partition_item** is the same as that of specifying the partition when creating a list partitioned table, except that the boundary value in the partition definition cannot be **DEFAULT**.
 - Except for the last partition, the boundaries of other partitions must be explicitly defined. The defined boundary cannot be **DEFAULT** and must be a non-empty proper subset of the source partition boundary. The boundary of the last partition is the difference set between the source partition boundary and other partition boundaries, and the boundary of the last partition is empty (that is, the difference set cannot be empty).
 - If the source partition is a **DEFAULT** partition, the boundary of the last partition is **DEFAULT**.
-

ADD PARTITION Sub-syntax: Adding One or More Partitions to a Specified Partitioned Table

add_clause specifies range partitioning.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
ADD { partition_less_than_item... | partition_start_end_item }
```

 CAUTION

- The **partition_less_than_item** syntax can only be used for range partitioned tables. Otherwise, an error will be reported.
- The syntax of **partition_less_than_item** is the same as the syntax specifying partitions in creating a range partitioned table.
- If the boundary value of the last partition is a MAXVALUE, new partitions cannot be added. Otherwise, an error will be reported.

add_clause specifies list partitioning.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
ADD list_partition_item
```

 CAUTION

- The **list_partition_item** syntax can only be used for a list partitioned table. Otherwise, an error will be reported.
- The **list_partition_item** syntax is the same as the syntax for specifying partitions when creating a list partitioned table.
- If the current partitioned table contains DEFAULT partitions, no new partitions can be added. Otherwise, an error will be reported.

DROP PARTITION Sub-syntax: Deleting a Specified Partition or Multiple Partitions from a Partitioned Table

Delete a specified partition.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
DROP PARTITION { partition_name | FOR ( partition_value [, ...] ) }
```

Delete multiple partitions. (This feature is supported by clusters of version 8.1.3.100 or later.)

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
DROP PARTITION { partition_name [, ...] }
```

TRUNCATE PARTITION: Clearing Data in a Table Partition

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
TRUNCATE PARTITION { partition_name | FOR ( partition_value [, ...] ) };
```

- **partition_value** indicates the partition key value. Multiple partition key values can be specified. Use commas (,) to separate multiple partition key values.
- When the **PARTITION FOR** clause is used, the entire partition where **partition_value** is located is cleared.

Syntax for Modifying a Partition Name

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
RENAME PARTITION { partition_name | FOR ( partition_value [, ...] ) } TO partition_new_name;
```

Table 21-167 Parameters for modifying a partition name

| Parameter | Description | Value Range |
|--------------------|--|---|
| partition_new_name | Specifies the new name of the partition. | A string compliant with the identifier naming rules . |

Example: Create a Range Partitioned Table `customer_address`

```
DROP TABLE IF EXISTS customer_address;
CREATE TABLE 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)
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE(ca_address_sk)
(
  PARTITION P1 VALUES LESS THAN(100),
  PARTITION P2 VALUES LESS THAN(200),
  PARTITION P3 VALUES LESS THAN(300)
);
```

Example: Create a List Partitioned Table

```
DROP TABLE IF EXISTS data_list;
CREATE TABLE data_list(
  id int,
  time int,
  salary decimal(12,2)
)PARTITION BY LIST (time)(
  PARTITION P1 VALUES (202209),
  PARTITION P2 VALUES (202210,202208),
  PARTITION P3 VALUES (202211),
  PARTITION P4 VALUES (202212),
  PARTITION P5 VALUES (202301)
);
```

Example: Use **MODIFY PARTITION** to Set Whether a Partitioned Index Is Usable

Create the local index **student_grade_index** for the partitioned table **customer_address** and set partition index names.

```
CREATE INDEX customer_address_index ON customer_address(ca_address_id) LOCAL
(
  PARTITION P1_index,
  PARTITION P2_index,
  PARTITION P3_index
);
```

Rebuild all indexes on partition **P1** in the partitioned table **customer_address**.

```
ALTER TABLE customer_address MODIFY PARTITION P1 REBUILD UNUSABLE LOCAL INDEXES;
```

Set all indexes in partition **P3** of the partitioned table **customer_address** to be unusable.

```
ALTER TABLE customer_address MODIFY PARTITION P3 UNUSABLE LOCAL INDEXES;
```

Example: Use REBUILD PARTITION to Rebuild a Partitioned Index into Which Data Has Been Inserted

Step 1 Creates a partitioned table.

```
DROP TABLE IF EXISTS sales;
CREATE TABLE sales (
  id      INT,
  sale_date DATE,
  amount  DECIMAL(10,2)
)
PARTITION BY RANGE (sale_date)
(
  PARTITION p202310 VALUES LESS THAN ('2023-11-01'),
  PARTITION p202311 VALUES LESS THAN ('2023-12-01')
);
```

Step 2 Create an index.

```
CREATE INDEX idx_sale_date ON sales (sale_date) LOCAL;
```

Step 3 Insert partition data of October and November.

```
INSERT INTO sales PARTITION (p202310)
VALUES
(1, '2023-10-05', 100.50),
(2, '2023-10-10', 200.75);
INSERT INTO sales PARTITION (p202311)
VALUES
(3, '2023-11-15', 300.00);
```

Step 4 Rebuild indexes of the specific partition **p202310**.

```
ALTER TABLE sales REBUILD PARTITION p202310 WITHOUT UNUSABLE;
```

----End

Example: Use ADD PARTITION to Add One or More Partitions to a Specified Partitioned Table

Add a partition to the range partitioned table **customer_address**.

```
ALTER TABLE customer_address ADD PARTITION P5 VALUES LESS THAN (500);
```

Add the following partitions to the range partitioned table **customer_address**:
[500, 600), [600, 700).

```
ALTER TABLE customer_address ADD PARTITION p6 START(500) END(700) EVERY(100);
```

Add the **MAXVALUE** partition **p7** to the range partitioned table **customer_address**.

```
ALTER TABLE customer_address ADD PARTITION p7 END(MAXVALUE);
```

Add partition **P6** to a list partitioned table.

```
ALTER TABLE data_list ADD PARTITION P6 VALUES (202302,202303);
```

Example: Use SPLIT PARTITION to Split a Partition into Multiple Partitions

Split partition **P7** in the range partitioned table **customer_address** at **800**.

```
ALTER TABLE customer_address SPLIT PARTITION P7 AT(800) INTO (PARTITION P6a,PARTITION P6b);
```

Split the partition at **400** in the range partitioned table **customer_address** into multiple partitions.

```
ALTER TABLE customer_address SPLIT PARTITION FOR(400) INTO(PARTITION p_part START(300) END(500) EVERY(100));
```

Split partition **P2** in the list partitioned table **data_list** into two partitions: **p2a** and **p2b**.

```
ALTER TABLE data_list SPLIT PARTITION P2 VALUES(202210) INTO (PARTITION p2a,PARTITION p2b);
```

Example: Use EXCHANGE PARTITION to Migrate Data from a Common Table to a Specified Partition

The following example demonstrates how to migrate data from table **math_grade** to partition **math** in partitioned table **student_grade**.

1. Create the partitioned table **student_grade**.

```
CREATE TABLE student_grade (  
    stu_name    char(5),  
    stu_no     integer,  
    grade      integer,  
    subject    varchar(30)  
)  
PARTITION BY LIST(subject)  
(  
    PARTITION gym VALUES('gymnastics'),  
    PARTITION phys VALUES('physics'),  
    PARTITION history VALUES('history'),  
    PARTITION math VALUES('math')  
);
```

2. Add data to the partitioned table **student_grade**.

```
INSERT INTO student_grade VALUES  
( 'Ann', 20220101, 75, 'gymnastics'),  
( 'Jack', 20220103, 60, 'math'),  
( 'Anna', 20220108, 56, 'history'),  
( 'Jann', 20220107, 82, 'physics'),  
( 'Molly', 20220104, 91, 'physics'),  
( 'Sam', 20220105, 72, 'math');
```

3. Query the records of partition **math** in **student_grade**.

```
SELECT * FROM student_grade PARTITION (math);
```

The query result is as follows:

| stu_name | stu_no | grade | subject |
|----------|----------|-------|---------|
| Jack | 20220103 | 60 | math |
| Sam | 20220105 | 72 | math |

(2 rows)

4. Create an ordinary table **math_grade** that matches the definition of the partitioned table **student_grade**.

```
CREATE TABLE math_grade  
(  
    stu_name    char(5),  
    stu_no     integer,  
    grade      integer,  
    subject    varchar(30)  
);
```

5. Insert data to the **math_grade** table. The data in the **student_grade** partitioned table conforms to the partition rule of partition **math**.

```
INSERT INTO math_grade VALUES  
( 'Ann', 20220101, 75, 'math'),  
( 'Jack', 20220103, 60, 'math'),
```

```
('Anna', 20220108, 56, 'math'),  
('Jann', 20220107, 82, 'math');
```

6. Migrate data from the ordinary table **math_grade** to partition **math** in the partitioned table **student_grade**.

```
ALTER TABLE student_grade EXCHANGE PARTITION (math) WITH TABLE math_grade;
```

7. Query the **student_grade** partitioned table. The results show that data in the **math_grade** table has exchanged with data in the **math** partition of the **student_grade** table.

```
SELECT * FROM student_grade PARTITION (math);  
stu_name | stu_no | grade | subject
```

```
-----+-----+-----+-----  
Anna    | 20220108 | 56 | math  
Jeck    | 20220103 | 60 | math  
Ann     | 20220101 | 75 | math  
Jann    | 20220107 | 82 | math  
(4 rows)
```

8. Query the **math_grade** table. The result shows that the data stored in the **math** partition of the **student_grade** partitioned table has been exchanged to the **math_grade** table.

```
SELECT * FROM math_grade;  
stu_name | stu_no | grade | subject
```

```
-----+-----+-----+-----  
Jeck    | 20220103 | 60 | math  
Sam     | 20220105 | 72 | math  
(2 rows)
```

Example: Use TRUNCATE PARTITION to Clear Data in a Table Partition

Clear the **p1** partition of the **customer_address** table.

```
ALTER TABLE customer_address TRUNCATE PARTITION p1;
```

Examples: Enabling and Disabling ROW MOVEMENT

If **ROW MOVEMENT** is enabled, the data of a row is moved to a new partition when the partition key of the row is updated to a new value and the new value belongs to another partition.

If a partition key is frequently updated, enabling **ROW MOVEMENT** may cause extra I/O overhead. To enable it, evaluate service requirements first.

Create a table partitioned by date and enable **ROW MOVEMENT**.

```
DROP TABLE IF EXISTS sale_data;  
CREATE TABLE sales_data (  
  sale_id INT,  
  product_name VARCHAR(100),  
  sale_date DATE,  
  amount DECIMAL(10, 2)  
)  
PARTITION BY RANGE (sale_date) (  
  PARTITION p_2023q1 VALUES LESS THAN ('2023-04-01'),  
  PARTITION p_2023q2 VALUES LESS THAN ('2023-07-01'),  
  PARTITION p_2023q3 VALUES LESS THAN ('2023-10-01'),  
  PARTITION p_2023q4 VALUES LESS THAN ('2024-01-01')  
)  
ENABLE ROW MOVEMENT;
```

Insert test data.

```
INSERT INTO sales_data (sale_id, product_name, sale_date, amount)  
VALUES
```

```
(1, 'Product A', '2023-02-15', 1000.00), -- It belongs to p_2023q1.  
(2, 'Product B', '2023-05-20', 1500.00); -- It belongs to p_2023q2.
```

Query the data distribution of each partition.

```
SELECT tableoid::regclass AS partition, * FROM sales_data ORDER BY sale_id;
```

| partition | sale_id | product_name | sale_date | amount |
|------------|---------|--------------|---------------------|---------|
| 2147617615 | 1 | Product A | 2023-02-15 00:00:00 | 1000.00 |
| 2147617616 | 2 | Product B | 2023-05-20 00:00:00 | 1500.00 |
| (2 rows) | | | | |

Update the partition key (which is moved from **p_2023q1** to **p_2023q2**).

```
UPDATE sales_data SET sale_date = '2023-05-01' WHERE sale_id = 1;
```

Query the data distribution of each partition again to confirm that the row has been moved.

```
SELECT tableoid::regclass AS partition, * FROM sales_data ORDER BY sale_id;
```

| partition | sale_id | product_name | sale_date | amount |
|------------|---------|--------------|---------------------|---------|
| 2147617616 | 1 | Product A | 2023-05-01 00:00:00 | 1000.00 |
| 2147617616 | 2 | Product B | 2023-05-20 00:00:00 | 1500.00 |
| (2 rows) | | | | |

Disable **ROW MOVEMENT**.

```
ALTER TABLE sales_data DISABLE ROW MOVEMENT;
```

Modify the partition key again (which is move from **p_2023q2** back to **p_2023q1**).

```
UPDATE sales_data SET sale_date = '2023-02-15' WHERE sale_id = 1;
```

View the command output. It indicates that **ROW MOVEMENT** is disabled and the partition key cannot be updated.

```
gaussdb=> UPDATE sales_data SET sale_date = '2023-02-15' WHERE sale_id = 1;  
ERROR:  dn_6003_6004: fail to update partitioned table "sales_data"  
DETAIL:  disable row movement
```

Example: Use **MERGE PARTITIONS** to Merge Multiple Partitions into One

Merge partitions **P2** and **P3** in the range partitioned table **customer_address** into one.

```
ALTER TABLE customer_address MERGE PARTITIONS P2, P3 INTO PARTITION P_M;
```

Example: Use **DROP PARTITION** to Delete a Specified Partition from a Partitioned Table

Delete partitions **P6a** and **P6b** from partitioned table **customer_address**.

```
ALTER TABLE customer_address DROP PARTITION P6a, P6b;
```

Helpful Links

[CREATE TABLE PARTITION](#) and [DROP TABLE](#)

21.13.4.6 RENAME TABLE

Function

RENAME TABLE renames a specified table.

RENAME TABLE has the same function as the following command:

```
ALTER TABLE table_name RENAME to new_table_name
```

Precautions

Tables cannot be renamed across schemas.

Syntax

```
RENAME TABLE  
{[schema.]table_name TO new_table_name} [, ...];
```

Parameter Description

Table 21-168 RENAME TABLE parameters

| Parameter | Description | Value Range |
|----------------|---|---|
| schema | Specifies the schema name. | Valid schema name |
| table_name | Specifies the name of the table to be modified. | Name of an existing table |
| new_table_name | Specifies the new table name. | A string that must comply with the rules in Identifier Naming Conventions . |

Examples

Create a column-store table and specify the storage format and compression mode.

```
DROP TABLE IF EXISTS customer_address;  
CREATE TABLE 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)  
)  
WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH,COLVERSION=2.0)  
DISTRIBUTE BY HASH (ca_address_sk);
```


Rename a table.

```
RENAME TABLE customer_address TO new_customer_address;
```

Helpful Links

[CREATE TABLE](#), [ALTER TABLE](#), [DROP TABLE](#)

21.13.4.7 DROP TABLE

Function

DROP TABLE deletes a specified table.

Precautions

- **DROP TABLE** forcibly deletes a specified table. After a table is deleted, any indexes that exist for the table will be deleted; any functions or stored procedures that use this table cannot be run. Deleting a partitioned table also deletes all partitions in the table.
- Only the table owner, schema owner, or a user granted with the DROP permission can run **DROP TABLE** on a table. A system administrator has this permission by default. To delete all the rows in a table but retain the table definition, use **TRUNCATE** or **DELETE**.



DANGER

- Be cautious when using **DROP OBJECT** (e.g., **DATABASE**, **USER/ROLE**, **SCHEMA**, **TABLE**, **VIEW**) as it may cause data loss, especially with **CASCADE** deletions. Always back up data before proceeding.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

Syntax

```
DROP TABLE [ IF EXISTS ]  
{ [schema.]table_name } [, ...] [ CASCADE | RESTRICT ] ;
```

Parameter Description

Table 21-169 DROP TABLE parameters

| Parameter | Description | Value Range |
|-----------|---|-------------|
| IF EXISTS | Reports a notice instead of an error if the specified table does not exist. | - |
| schema | Name of the schema to which the table to be deleted belongs. | - |

| Parameter | Description | Value Range |
|--------------------|---|--|
| table_name | Specifies the name of the table to be deleted. | An existing table name. |
| CASCADE RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none">• CASCADE: automatically deletes objects (such as views) that depend on the table to be deleted.• RESTRICT (default): refuses to delete the table if any objects depend on it. This is the default. |
| PURGE | If the Purge parameter is specified, data is directly deleted regardless of whether the recycle bin feature is enabled. This is supported only by clusters of version 9.1.0.200 or later. | - |
| KEEP ttl
HOUR | <p>In V3 tables, enabling the recycle bin feature and setting a TTL allows physical files to be moved to the recycle bin for potential restoration. However, once the TTL expires, the background process automatically clears the objects in the recycle bin. This is supported only by clusters of version 9.1.0.200 or later.</p> <p>NOTICE
You are advised to use this parameter in scenarios where the DROP and TRUNCATE operations are not frequently performed.</p> | - |

Examples

Delete **warehouse_t1** table.

```
DROP TABLE tpcds.warehouse_t1;
```

Helpful Links

[ALTER TABLE](#), [12.101-RENAME TABLE](#), and [CREATE TABLE](#)

21.13.4.8 CREATE FOREIGN TABLE (SQL on OBS or Hadoop)

Function

CREATE FOREIGN TABLE creates an HDFS or OBS foreign table in the current database to access structured data stored in the HDFS or OBS distributed cluster

file system. You can also use the HDFS or OBS foreign table to export data in ORC and Parquet formats to HDFS or OBS.

A foreign table is a virtual table within a database or big data platform that does not physically store data. Instead, it serves as a mapping to data residing in external storage systems, such as HDFS or OBS. This mapping is established using metadata, which includes the table structure, the location of the data, and its file format. It enables users to directly access or operate external data using standard SQL statements. A foreign table serves as a data proxy that logically maps data in external file systems to table structures that can be identified by databases.

Foreign tables are designed to break data silos. Foreign tables allow database users to access external system data in a familiar way through metadata mapping and export data to external systems. In this way, data can be efficiently collaborated, analyzed, and shared across systems, especially in big data scenarios, significantly reducing data movement costs and improving collaboration efficiency.

When DWS accesses external data sources through foreign tables, the access performance varies according to data sources.

- Data stored in OBS: Data storage is decoupled from compute. The cluster storage cost is low, and storage capacity is not limited. Clusters can be deleted at any time. However, the compute performance depends on the OBS access performance and is lower than that of HDFS. OBS is recommended when data computing is infrequent.
- Data stored in HDFS: Data storage is not decoupled from compute. The cluster storage cost is high, and storage capacity is limited. The compute performance is high. You must export data before you delete clusters. HDFS is recommended for computing-intensive scenarios.

NOTE

The following read and write foreign tables are supported only in the storage-compute decoupling 3.0 version.

Precautions

- When using an OBS foreign table to access data in an OBS bucket, you need to ensure that your DWS cluster and OBS bucket are in the same region.
- Only the system administrator **dbadmin** or a common user who has been granted the USEFT permission can perform foreign table operations.

```
ALTER USER user_name USEFT;
```
- The HDFS foreign table and OBS foreign table are classified into read-only, write-only, and read-write foreign tables. Read-only foreign tables are used for query. Write-only foreign tables can be used to export data from DWS to the distributed file system. Read-write foreign tables can be used for query and data export. Only 9.1.0.100 and later versions support read-write foreign tables.
- Supported formats:
 - Importing data: Only data in ORC, TEXT, CSV, CARBONDATA, PARQUET, and JSON formats can be imported for query.
 - Exporting data: For OBS foreign tables, data in ORC and PARQUET (supported only in storage-compute decoupling 3.0 version) can be exported. For HDFS foreign tables, data in ORC and PARQUET can be exported.

- Before using a foreign table, you need to manually create a foreign server, which is a virtual object configured in the database and used to define parameters (such as the address, protocol, and authentication information) for connecting to an external data source. Through the foreign servers, the local database can directly query and insert external data, just like operating local tables. (The syntax is similar, but may be restricted by performance or functions.) For details, see [CREATE SERVER](#).
- If the foreign data wrapper is set to **HDFS_FDW** or **DFS_FDW** when you manually create a foreign server, you need to configure the **DISTRIBUTE BY** clause to specify the distribution mode when creating a read-only foreign table.

Table 21-170 Read and write formats supported by OBS foreign tables

| Data Type | DFS_FDW/HDFS_FDW | | |
|------------|------------------|------------|------------|
| | READ ONLY | WRITE ONLY | READ WRITE |
| - | | | |
| ORC | √ | √ | √ |
| PARQUET | √ | √ | √ |
| CARBONDATA | √ | × | × |
| TEXT | √ | × | × |
| CSV | √ | × | × |
| JSON | √ | × | × |

Syntax

Create an HDFS foreign table.

```
CREATE FOREIGN TABLE [ IF NOT EXISTS ] table_name
( [ { column_name type_name
  [ { [CONSTRAINT constraint_name] NULL |
    [CONSTRAINT constraint_name] NOT NULL |
    column_constraint [...] } ] |
  table_constraint [, ...] [, ...] ] )
SERVER server_name
OPTIONS ( { option_name 'value ' [, ...] )
[ {WRITE ONLY | READ ONLY | READ WRITE} ]
DISTRIBUTE BY {ROUNDROBIN | REPLICATION}

[ PARTITION BY ( column_name ) [ AUTOMAPPED ] ] ;
```

- **column_constraint** is as follows:
[CONSTRAINT constraint_name]
{PRIMARY KEY | UNIQUE}
[NOT ENFORCED [ENABLE QUERY OPTIMIZATION | DISABLE QUERY OPTIMIZATION] | ENFORCED]
- **table_constraint** is as follows:
[CONSTRAINT constraint_name]
{PRIMARY KEY | UNIQUE} (column_name)
[NOT ENFORCED [ENABLE QUERY OPTIMIZATION | DISABLE QUERY OPTIMIZATION] | ENFORCED]

Parameter Description

Table 21-171 CREATE FOREIGN TABLE (SQL on OBS or Hadoop) parameters

| Parameter | Description | Value Range |
|---------------|---|--|
| IF NOT EXISTS | Sends a notice, but does not throw an error, if a table with the same name exists. | - |
| table_name | Specifies the name of the foreign table. | A string, which must comply with the naming convention. For details, see Identifier Naming Conventions . |
| column_name | Specifies the name of a column in the foreign table. Columns are separated by commas (,). | <p>A string, which must comply with the naming convention. For details, see Identifier Naming Conventions.</p> <p>A JSON object consists of nested or parallel name-value pairs, which are irrelevant to the sequence. When data in JSON format is imported, the mapping between fields and values is determined based on the automatic mapping between field names and names of name-value pairs. You need to define proper field names. Otherwise, you may not get the expected result. The rules for automatic mapping between field names and names of name-value pairs are as follows:</p> <ul style="list-style-type: none">• If there are no nesting or arrays, the field names must be the same as the names of name-value pairs, case insensitive.• Use underscores (_) to concatenate two names to identify the nesting relationship.• A field name uses the number sign (#) and a decimal non-negative integer (n) to identify the nth element (starting from 0) of an array. <p>For example, to import each element of the <code>{"A": "simple", "B": {"C": "nesting"}, "D": ["array", 2, {"E": "complicated"}]}</code> object, the field names in the foreign table must be defined as <code>a</code>, <code>b</code>, <code>b_c</code>, <code>d</code>, <code>d#0</code>, <code>d#1</code>, <code>d#2</code> and <code>d#2_e</code>. The sequence in which the fields are defined does not affect the import result.</p> |
| type_name | Specifies the data type of the column. | - |

| Parameter | Description | Value Range |
|---------------------|--|--|
| constraint_name | Specifies the name of a constraint for the foreign table. | - |
| { NULL NOT NULL } | <p>Specifies whether the column allows NULL.</p> <p>When you create a table, whether the data in HDFS is NULL or NOT NULL cannot be guaranteed. The consistency of data is guaranteed by users. Users must decide whether the column is NULL or NOT NULL. (The optimizer optimizes the NULL/NOT NULL and generates a better plan.)</p> | - |
| SERVER server_name | Specifies the server name of the foreign table. Users can customize its name. | A string indicating an existing server. It must comply with the naming convention. |

| Parameter | Description | Value Range |
|-------------------------------------|--|---|
| OPTIONS | Specifies parameters of foreign table data. | <p>The OPTIONS parameters to be specified vary depending on the scenario. The options are as follows:</p> <ul style="list-style-type: none">• Data format parameters. For details, see Table 21-172.• Error-tolerance parameters. For details, see Table 21-173.• Parameters that need to be specified only when the server type is DLI. For details, see Table 21-174.• Parameters that need to be specified only when format is set to hudi. For details, see OPTIONS parameters (only when format is set to hudi).• OBS foreign table. Different data source formats support different OPTIONS parameters. For details, see Table 21-176.• HDFS foreign table. Different data source formats support different OPTIONS parameters. For details, see Table 21-177. |
| WRITE ONLY READ ONLY READ WRITE | Specifies the type of the foreign table to be created. | <ul style="list-style-type: none">• WRITE ONLY creates a write-only HDFS/OBS foreign table.• READ ONLY creates a read-only HDFS/OBS foreign table.• READ WRITE creates a read/write HDFS/OBS foreign table. <p>If the foreign table type is not specified, a read-only foreign table is created by default.</p> |
| DISTRIBUTE BY ROUND ROBIN | Specifies ROUNDROBIN as the distribution mode for the HDFS/OBS foreign table. Each row of the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. | - |

| Parameter | Description | Value Range |
|---------------------------|--|-------------|
| DISTRIBUTE BY REPLICATION | Specifies REPLICATION as the distribution mode for the HDFS foreign table. Each row in the table exists on all DN. That is, full table data is stored on each DN. | - |

| Parameter | Description | Value Range |
|---------------------------------------|--|--|
| PARTITION BY (column_name) AUTOMAPPED | column_name specifies the partition column. AUTOMAPPED means the partition column specified by the HDFS partitioned foreign table is automatically mapped with the partition directory information in HDFS. The prerequisite is that the sequences of partition columns specified in the HDFS foreign table and in the directory are the same. This function is applicable only to read-only foreign tables. | <ul style="list-style-type: none">• A write-only partitioned foreign table can have up to four levels of partitions, meaning that you can specify a maximum of four columns as partition columns. At least one column must be reserved as a non-partition column.• HDFS read-only foreign tables support partition tables in text, CSV, CarbonData, ORC, and Parquet formats.• HDFS write-only foreign tables support ORC and Parquet partitioned tables.• HDFS read and write foreign tables support ORC and Parquet partitioned tables.• OBS read-only, write-only, and read-write foreign tables support ORC and Parquet partitioned tables.• Columns of the floating point or Boolean type cannot be used as partition columns.• The maximum length of a partition field can be specified by the GUC parameter dfs_partition_directory_length.• A partition directory name is in the format <i>Partition column name=Partition column value</i>. Special characters in the name will be escaped. To ensure that the total length of the name after escaping does not exceed dfs_partition_directory_length, it is advisable to keep the name length before escaping less than or equal to $(\text{dfs_partition_directory_length} + 1)/3$.• Do not use a column containing too many Chinese characters as a partition column. You may encounter errors when calculating the final partition directory name's length due to the different space requirements of Chinese and English characters. This is particularly true when the partition directory name exceeds the dfs_partition_directory_length length limit. |
| CONSTRAINT constraint_name | Specifies the name of informational constraint of the foreign table. | A string, which must comply with the naming convention. |

| Parameter | Description | Value Range |
|-------------|--|-------------|
| PRIMARY KEY | The primary key constraint specifies that one or more columns of a table must contain unique (non-duplicate) and non-null values. Only one primary key can be specified for a table. | - |
| UNIQUE | Specifies that a group of one or more columns of a table must contain unique values. For the purpose of a unique constraint, NULL is not considered equal. | - |

| Parameter | Description | Value Range |
|--------------|---|--|
| NOT ENFORCED | <p>Specifies that the created constraint is an informational constraint, which is not forcibly verified by the database. This parameter is used with ENABLE QUERY OPTIMIZATION.</p> <ul style="list-style-type: none">• Informational rather than mandatory: You still need to specify a unique constraint for a column using PRIMARY KEY or UNIQUE to optimize the query execution plan.• No mandatory verification: The database does not check whether the data in the external data source actually meets constraint conditions (for example, whether duplicate values exist).• Performance optimization: The query optimizer is allowed to use an informational constraint to generate a more efficient execution plan. However, you need to ensure | <p>For example, the primary key is declared, but the verification is not mandatory.</p> <pre>CREATE FOREIGN TABLE hdfs_users (user_id INT PRIMARY KEY NOT ENFORCED,
...</pre> |

| Parameter | Description | Value Range |
|---------------------------|--|---|
| | <p>that the data meets constraint conditions, or the query result may be incorrect.</p> <ul style="list-style-type: none">• Risk: If the actual data violates constraint conditions (for example, duplicate values exist), the query result may be incorrect (for example, the aggregation result is abnormal or redundant records are generated during JOIN operations). <p>For more information, see Informational Constraints.</p> | |
| ENFORCED | Specifies that the created constraint is an enforced constraint that is forcibly validated by the database. This parameter is reserved. Currently, DWS does not support ENFORCED . | - |
| PRIMARY KEY (column_name) | Specifies the informational constraint on column_name . | A string indicating an existing column_name. It must comply with the naming convention. |

| Parameter | Description | Value Range |
|----------------------------|--|--|
| ENABLE QUERY OPTIMIZATION | Enables the query optimizer to use informational constraints to generate a more efficient execution plan. This parameter is used together with NOT ENFORCED . | Create a foreign table, add a unique constraint to the p_partkey int column, do not enforce the constraint, and allow the optimizer to use the informational constraint to generate a better execution plan.
CREATE FOREIGN TABLE ft_part
(
p_partkey int UNIQUE NOT ENFORCED ENABLE QUERY OPTIMIZATION,
...
) |
| DISABLE QUERY OPTIMIZATION | Disables the query optimizer to use informational constraints to generate an efficient execution plan. | - |

Table 21-172 OPTIONS parameters of the data format

| Parameter | Description | Value Range |
|-----------|---|---|
| header | Specifies whether a data file contains a table header. header is available only for CSV files. | <ul style="list-style-type: none">• true or on: The first row of the data file will be identified as the header and ignored during export.• false or off: The first row is identified as data. The default value is false or off . |
| quote | Specifies the quotation mark for the CSV format. The default value is a double quotation mark ("). | The quote value cannot be the same as the delimiter or null value.
The quote value must be a single-byte character.
Invisible characters are recommended as quote values, such as 0x07, 0x08, and 0x1b. |

| Parameter | Description | Value Range |
|-----------|---|-------------|
| escape | <p>Specifies an escape character for a CSV file. The value must be a single-byte character.</p> <p>In a database, an escape character is a special character used to escape other characters. It enables the database system to identify and correctly process characters that have special meanings and may conflict with SQL syntax (such as quotation marks and special characters) as common characters. Common escape characters include backslashes (\) and double quotation marks (").</p> <p>The default value is a double quotation mark ("). If it is the same as the value of quote, it will be replaced with \0.</p> | - |

| Parameter | Description | Value Range |
|-----------|---|---|
| location | <p>Specifies the file path on OBS. This is an OBS foreign table parameter. The data sources of multiple buckets are separated by vertical bars (), for example, LOCATION 'obs://bucket1/folder/ obs://bucket2/'.</p> <p>The database scans all objects in the specified folders.</p> <p>When accessing a DLI multi-version table, you do not need to specify the location parameter.</p> | - |
| format | <p>Specifies the format of the source data file in a foreign table.</p> | <p>The HDFS foreign table can read data in ORC, TEXT, JSON, CSV, Parquet, and Hudi file formats when in READ ONLY mode. But if you are using the WRITE ONLY/READ WRITE foreign table, you can only work with data in ORC and Parquet file formats.</p> <p>OBS foreign tables allow you to read data in ORC, TEXT, JSON, CSV, CarbonData, Parquet, and Hudi file formats in read-only mode. But if you are using write-only mode, you can only work with data in ORC and Parquet file formats.</p> <p>CAUTION</p> <ul style="list-style-type: none">Only JSON objects (embraced in {}) can be imported. JSON arrays (embraced in []) cannot be imported. However, arrays inside a JSON object can be imported.Apache Hudi is a transactional data lake platform. In the current version, you can set the format parameter of the OBS foreign table to Hudi to access the corresponding Hudi table data sets. This version also provides the method of accessing the incremental or real-time view of the Hudi table. |

| Parameter | Description | Value Range |
|------------|--|---|
| foldername | <p>The directory of the data source file in the foreign table, that is, the corresponding file directory in HDFS or on OBS. This is mandatory for WRITE ONLY and READ WRITE writable foreign tables and optional for READ ONLY foreign tables.</p> <p>When accessing a DLI multi-version table, you do not need to specify the foldername parameter.</p> <ul style="list-style-type: none">• OBS foreign table. The format is <i>/OBS bucket name/folder name/</i>. Ensure that the OBS bucket and DWS cluster are in the same region. Cross-region access to OBS bucket data is not supported.• HDFS foreign table. The format is the directory where data files in the remote HDFS are stored. For an MRS cluster, log in to the MRS console, go to the cluster basic information page, click Files, select HDFS File List, and find the | <p>Example of an OBS foreign table</p> <pre>OPTIONS (
 foldername '/obs-demo01/obs-dws/',
 format 'parquet',
 encoding 'utf8')</pre> <p>Example of an HDFS foreign table:</p> <pre>OPTIONS (
 foldername '/user/hive/warehouse/demo.db/
 product_info_orc/',
 format 'orc',
 encoding 'utf8')</pre> |

| Parameter | Description | Value Range |
|-----------|---|-------------|
| | directory where data is stored. | |
| encoding | It specifies the encoding format of a source data file in the foreign table. Its default value is utf8 . This parameter is optional. | - |
| totalrows | (Optional)
Estimated number of rows in a table. This parameter is used only for OBS foreign tables. Because OBS may store many files, it is slow to analyze data. This parameter allows you to set an estimated value so that the optimizer can estimate the table size according to the value. Generally, query efficiency is high when the estimated value is close to the actual value. | - |

| Parameter | Description | Value Range |
|-----------|--|---|
| filenames | Data source files specified in the foreign table. Multiple files are separated by commas (,). | <ul style="list-style-type: none">You are advised to use the foldername parameter to specify the location of the data source. For a read-only foreign table, either filenames or foldername must be specified. For a write-only foreign table or read-write foreign table, only foldername can be specified.If foldername is an absolute directory, it should be enclosed by slashes (/). Multiple paths are separated by commas (,).When you query a partitioned table, data is pruned based on partition information, and data files that meet the requirement are queried. Pruning involves scanning HDFS directory contents many times. Therefore, do not use columns with low repetition as partition column.OBS read-only foreign tables and read/write foreign tables are not supported. |
| delimiter | Specifies the column delimiter of data, and uses the default delimiter if it is not set. The default delimiter of TEXT is a tab. | <p>The value of delimiter can be a multi-character delimiter whose length is less than or equal to 10 bytes.</p> <ul style="list-style-type: none">A delimiter cannot be \r or \n.A delimiter cannot be the same as the null parameter.A separator cannot contain (\), (.), digits, or letters.The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.You are advised to use a multi-character, such as the combination of the dollar sign (\$), caret (^), ampersand (&), or invisible characters, such as 0x07, 0x08, and 0x1b as the delimiter.delimiter is available only for TEXT and CSV source data files. |

| Parameter | Description | Value Range |
|-------------|---|---|
| eol | Specifies the newline character style of the imported data file. | <p>Multi-character newline characters are supported, but the newline character cannot exceed 10 bytes. Common newline characters include \r (0x0D), \n (0x0A), and \r\n (0x0D0A). Special newline characters include \$ and #.</p> <ul style="list-style-type: none">• The eol parameter applies only to TEXT files.• The value of the eol parameter cannot be the same as that of DELIMITER or NULL.• The value of the eol parameter cannot contain digits, letters, or periods (.). |
| null | Specifies the string that represents a null value. | <p>The default value is \N for the TEXT format.</p> <ul style="list-style-type: none">• The null value cannot be \r or \n. The maximum length is 100 characters.• The null parameter cannot be the same as the delimiter.• null is available only for TEXT and CSV source data files. |
| noescaping | Specifies in TEXT format, whether to escape the backslash (\) and its following characters. | <p>noescaping is available only for the TEXT format. true/on or false/off. The default value is false or off.</p> |
| date_format | Specifies the DATE format for data import. This parameter is valid only for READ ONLY foreign tables. | <p>The value must be in a valid DATE format. For details, see Date and Time Processing Functions and Operators.</p> <ul style="list-style-type: none">• If ORACLE is specified as the compatible database, the DATE format is TIMESTAMP. For details, see timestamp_format below.• date_format is available only for TEXT and CSV source data files. |
| time_format | Specifies the TIME format for data import. This parameter is valid only for READ ONLY foreign tables. | <p>This parameter is available only for TEXT and CSV source data files.</p> <p>Value range: Valid TIME. Time zones cannot be used. For details, see Date and Time Processing Functions and Operators.</p> |

| Parameter | Description | Value Range |
|--------------------------|--|---|
| timestamp_format | Specifies the TIMESTAMP format for data import. This parameter is valid only for READ ONLY foreign tables. | This parameter is available only for TEXT and CSV source data files.
Value range: any valid TIMESTAMP value. Time zones are not supported. For details, see Date and Time Processing Functions and Operators . |
| smalldatetimetime_format | Specifies the SMALLDATETIME format for data import. This parameter is valid only for READ ONLY foreign tables. | This parameter is available only for TEXT and CSV source data files.
Value range: any valid SMALLDATETIME value. For details, see Date and Time Processing Functions and Operators . |
| dataencoding | This parameter specifies the data code of the data table to be exported when the database code is different from the data code of the data table. For example, the database code is Latin-1, but the data in the exported data table is in UTF-8 format. This parameter is optional. If this parameter is not specified, the database encoding format is used by default. This syntax is valid only for the WRITE ONLY/READ WRITE foreign table of HDFS. | The parameter is available only for HDFS foreign tables in ORC format with WRITE ONLY or READ WRITE.
Value range: data code types supported by the database encoding |

| Parameter | Description | Value Range |
|-------------|---|--|
| filesize | Specifies the file size of a write-only foreign table. This parameter is optional. If this parameter is not specified, the file size in the distributed file system configuration is used by default. This syntax is valid only for WRITE ONLY/READ WRITE writable foreign tables. | The parameter is available only for HDFS foreign tables in ORC format with WRITE ONLY or READ WRITE.
Value range: an integer ranging from 1 to 1024 |
| compression | Specifies the file compression mode. This parameter is optional and is supported only by cluster versions 8.2.0 and later.

For a write only/read-write foreign table, it specifies the compression mode for ORC files.

For a READ ONLY foreign table, it specifies the compression mode for TEXT, CSV, or JSON files. | <ul style="list-style-type: none">For ORC files: the value can be zlib (default), snappy, and lz4.For Parquet files: zlib, snappy (default), lz4, lz4_hadoop.For TEXT, CSV, and JSON files: the value is gzip. CAUTION <ul style="list-style-type: none">In versions earlier than 8.2.0, the parameter cannot be specified for READ ONLY foreign tables.The lz4_hadoop compression option is designed to work with the Hive LZ4 algorithm. By setting the compression format for a parquet foreign table, both the parquet foreign table and Hive can seamlessly exchange data. |
| version | Specifies the ORC version number. This parameter is optional. This syntax is available only for WRITE ONLY/READ WRITE foreign tables. | Value range: Only 0.12 is supported. The default value is 0.12 . |

| Parameter | Description | Value Range |
|---------------|---|---|
| julian_adjust | <p>Specifies whether to correct the Julian day in the data. This parameter is optional and is supported only by cluster versions 8.3.0 and later.</p> <p>Spark 2.X uses the Julian day, and Spark 3.0 and later versions use the Proleptic Gregorian calendar. As a result, some dates before parquet 1582 are different. Correct the dates based on the site requirements.</p> | The value can be true or false . Default value: true |
| cache_policy | <p>Specifies the disk cache policy for foreign tables. This parameter is supported only in the storage-compute decoupling 3.0 version.</p> | <p>Value range: For foreign tables, this parameter can be set to ALL or NONE. ALL indicates that the hot cache in the disk cache is used, and NONE indicates that the cold cache is used. Hot cache occupies more space than cold cache and uses more complex replacement algorithms.</p> |

Table 21-173 OPTIONS fault tolerance parameters

| Parameter | Description | Value Range |
|---------------------|---|---|
| fill_missing_fields | Specifies whether to generate an error message when the last column in a row in the source file is lost during data loading. | true/on or false/off . The default value is false or off . <ul style="list-style-type: none">If this parameter is set to true or on and the last column of a data row in a data source file is lost, the column is replaced with NULL and no error message will be generated.If this parameter is set to false or off and the last column is missing, the following error information will be displayed:
missing data for column "tt" CAUTION <ul style="list-style-type: none">Because SELECT COUNT(*) does not parse columns in text format, it does not report missing columns.fill_missing_fields is available only for TEXT and CSV source data files. |
| ignore_extra_data | Specifies whether to ignore excessive columns when the number of data source files exceeds the number of foreign table columns. This parameter is available only during data importing. | true/on or false/off . The default value is false or off . <ul style="list-style-type: none">If this parameter is set to true or on and the number of data source files exceeds the number of foreign table columns, excessive columns will be ignored.If this parameter is set to false or off and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:
extra data after last expected column CAUTION <ul style="list-style-type: none">If the newline character at the end of the row is lost, setting the parameter to true will ignore data in the next row.Because SELECT COUNT(*) does not parse columns in text format, it does not report missing columns.ignore_extra_data is available only for TEXT and CSV source data files. |

| Parameter | Description | Value Range |
|---------------|--|--|
| checkencoding | Specifies whether to check the character encoding. | <p>Value range: no, low, and high The default value is low.</p> <p>In TEXT format, the rule of error tolerance for invalid characters imported is as follows:</p> <ul style="list-style-type: none">• \0 is converted to a space.• Other invalid characters are converted to question marks.• Setting checkencoding to low enables invalid characters toleration. If NULL and DELIMITER are set to spaces or question marks (?), errors like "illegal chars conversion may confuse null 0x20" will be displayed, prompting you to modify parameters that may cause confusion and preventing importing errors. <p>In ORC format, the rule of error tolerance for invalid characters imported is as follows:</p> <ul style="list-style-type: none">• If checkencoding is set to no, invalid characters are not checked during data import. If you confirm that there are no invalid characters, this option can shorten the import time.• If checkencoding is low, an imported field containing invalid characters will be replaced with a quotation mark string of the same length.• If checkencoding is high, data import stops when an invalid character is detected. |

| Parameter | Description | Value Range |
|---------------|---|---|
| force_mapping | Indicates the handling method used when no correct name-value pairs are matched for the foreign table columns in JSON format. | <p>The options include true and false. Default value: true</p> <ul style="list-style-type: none">• If force_mapping is true, null is entered in the corresponding column. The meaning of null is the same as that defined in JSON.• If force_mapping is false, an error is reported, indicating that the column does not exist. <p>CAUTION
There are no restrictions on JSON objects. While the definition of foreign table fields must comply with DWS identifier specifications (such as length and character restrictions). This import method may cause exceptions. For example, a JSON name cannot be correctly identified or a field is repeatedly defined. You are advised to use the fault tolerance option force_mapping or JSON operators (for details, see JSON/JSONB Functions and Operators).</p> <p>For JSON format, SELECT COUNT(*) does not parse specific fields. Therefore, no error is reported when a field is missing or the format is incorrect.</p> |

Table 21-174 OPTIONS parameters (only when the server type is DLI)

| Parameter | Description | Value Range |
|-------------------|---|-------------|
| dli_project_id | Specifies the project ID corresponding to DLI. You can obtain the project ID from the management console. This parameter is available only when the server type is DLI. This parameter is supported only in 8.1.1 or later. | - |
| dli_database_name | Specifies the name of the database where the DLI multi-version table to be accessed is located. This parameter is available only when the server type is DLI. This parameter is supported only in 8.1.1 or later. | - |
| dli_table_name | Specifies the name of the DLI multi-version table to be accessed. This parameter is available only when the server type is DLI. This parameter is supported only in 8.1.1 or later. | - |

Table 21-175 OPTIONS parameters (available only when **format** is set to **hudi**. The following parameters are supported only in clusters of version 9.1.0.100 or later.)

| Parameter | Description | Value Range |
|-------------|--|---|
| auth_server | Specifies a server, which is used for user identity authentication when KMS is used to encrypt data in the Hudi foreign table. This parameter can be specified only when format is set to Hudi . | Value range: name of an available HDFS server.

MRS provides the column-level encryption capability for Hudi data. RangerKMS is used to manage users and key IDs and control encryption and decryption permissions. You can specify the auth_server parameter when creating a foreign table to bind user RangerKMS to the table. In this way, you can be authorized and obtain keys to read encrypted data. |
| kms_url | Specifies the IP address and port number of the KMS service. This parameter can be specified only when format is set to hudi and auth_server is specified. | Value range: a valid string in ip:port format.

Generally, if auth_server is specified, the IP address and port number of the KMS service can be automatically parsed. If this parameter is specified, the address is forcibly used to access the KMS service. |
| cow_improve | Optimization parameter for the COPY_ON_WRITE table. When this parameter is enabled, DNs replace the dws-bigdata component to directly read data from OBS.

This parameter can be specified only when format is set to Hudi and cannot be specified together with auth_server . | Value range: true/on and false/off .
The default value is false or off .
NOTE
This parameter is invalid for MERGE_ON_READ tables. |

Table 21-176 OPTIONS supported by OBS foreign tables

| Parameter | OBS | | | | | | | | | | |
|---------------------|-----------|-----------|-----------|-----------|------------|-----------|------------|-----------|------------|-----------|-----------|
| | TEXT | CSV | JSON | ORC | | | CARBONDATA | PARQUET | | | HUDI |
| | READ ONLY | READ ONLY | READ ONLY | READ ONLY | WRITE ONLY | READWRITE | READ ONLY | READ ONLY | WRITE ONLY | READWRITE | READ ONLY |
| location | √ | √ | √ | √ | × | × | √ | √ | × | × | × |
| format | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| header | × | √ | × | × | × | × | × | × | × | × | × |
| delimiter | √ | √ | × | × | × | × | × | × | × | × | × |
| quote | × | √ | × | × | × | × | × | × | × | × | × |
| escape | × | √ | × | × | × | × | × | × | × | × | × |
| null | √ | √ | × | × | × | × | × | × | × | × | × |
| noescaping | √ | × | × | × | × | × | × | × | × | × | × |
| encoding | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| fill_missing_fields | √ | √ | × | × | × | × | × | × | × | × | × |
| ignore_extra_data | √ | √ | × | × | × | × | × | × | × | × | × |
| date_format | √ | √ | √ | × | × | × | × | × | × | × | × |

| Parameter | OBS | | | | | | | | | | |
|----------------------|-----|---|---|---|---|---|---|---|---|---|---|
| time_format | √ | √ | √ | × | × | × | × | × | × | × | × |
| timestamp_format | √ | √ | √ | × | × | × | × | × | × | × | × |
| smalldatetime_format | √ | √ | √ | × | × | × | × | × | × | × | × |
| chunksizes | √ | √ | √ | × | × | × | × | × | × | × | × |
| filenames | × | × | × | × | × | × | × | × | × | × | × |
| foldername | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| dataencoding | × | × | × | × | × | × | × | × | × | × | × |
| filesize | × | × | × | × | × | × | × | × | × | × | × |
| compression | √ | √ | √ | × | √ | √ | × | × | √ | √ | × |
| version | × | × | × | × | √ | √ | × | × | × | × | × |
| checkencoding | √ | √ | √ | √ | × | √ | √ | √ | √ | √ | √ |
| total rows | √ | √ | √ | √ | × | √ | × | × | × | × | × |

| Parameter | OBS | | | | | | | | | | |
|-----------------------|-----|---|---|---|---|---|---|---|---|---|---|
| force_mappi
ng | x | x | √ | x | x | x | x | x | x | x | x |
| auth_
server | x | x | x | x | x | x | x | x | x | x | √ |
| kms_
url | x | x | x | x | x | x | x | x | x | x | √ |
| cow_
impro
ve | x | x | x | x | x | x | x | x | x | x | √ |
| julia
n_ad
just | x | x | x | x | x | x | x | √ | x | x | √ |

Table 21-177 OPTIONS supported by HDFS foreign tables

| Parameter | HDFS | | | | | | | | |
|---------------|--------------|--------------|--------------|--------------|-------------------|-----------------------|--------------|-------------------|-----------------------|
| - | TEXT | CSV | JSON | ORC | | | PARQUET | | |
| | READ
ONLY | READ
ONLY | READ
ONLY | READ
ONLY | WRIT
E
ONLY | REA
D
WRIT
E | READ
ONLY | WRIT
E
ONLY | REA
D
WRI
TE |
| locati
on | x | x | x | x | x | x | x | x | x |
| forma
t | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| heade
r | x | √ | x | x | x | x | x | x | x |
| delimi
ter | √ | √ | x | x | x | x | x | x | x |
| quote | x | √ | x | x | x | x | x | x | x |
| escap
e | x | √ | x | x | x | x | x | x | x |
| null | √ | √ | x | x | x | x | x | x | x |

| Parameter | HDFS | | | | | | | | |
|-----------------------|------|---|---|---|---|---|---|---|---|
| noescaping | √ | × | × | × | × | × | × | × | × |
| encoding | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| fill_missing_fields | √ | √ | × | × | × | × | × | × | × |
| ignore_extra_data | √ | √ | × | × | × | × | × | × | × |
| date_format | √ | √ | √ | × | × | × | × | × | × |
| time_format | √ | √ | √ | × | × | × | × | × | × |
| timestamp_format | √ | √ | √ | × | × | × | × | × | × |
| smalldate_time_format | √ | √ | √ | × | × | × | × | × | × |
| chunk size | √ | √ | √ | × | × | × | × | × | × |
| filenames | √ | √ | √ | √ | × | × | √ | × | × |
| folder name | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| dataencoding | × | × | × | × | √ | √ | × | × | × |
| filesize | × | × | × | × | √ | √ | × | × | × |
| compression | √ | √ | √ | × | √ | √ | × | √ | √ |
| version | × | × | × | × | √ | √ | × | × | × |
| check encoding | √ | √ | √ | √ | √ | √ | √ | √ | √ |

| Parameter | HDFS | | | | | | | | |
|-----------------------|---------------|---|---|---|---|---|---|---|---|
| | totalr
ows | | | | | | | | |
| totalr
ows | x | x | x | x | x | x | x | x | x |
| force_
mappi
ng | x | x | √ | x | x | x | x | x | x |
| julian_
adjust | x | x | x | x | x | x | √ | x | x |

 NOTE

chunksize indicates the cache size of each OBS reading thread on a DN. The format can be text, CSV, or JSON. The default size is 4 MB.

Informational Constraints

In DWS, the use of data constraints depend on users. If users can make data sources strictly comply with certain constraints, the query on data with such constraints can be accelerated. Foreign tables do not support Index. Informational constraint is used for optimizing query plans.

The constraints of creating informational constraints for an HDFS foreign table are as follows:

- You can create an informational constraint only if the values in a NOT NULL column in your table are unique. Otherwise, the query result will be different from expected.
- Currently, the informational constraint of DWS supports only PRIMARY KEY and UNIQUE constraints.
- The informational constraints of DWS support the NOT ENFORCED attribute.
- UNIQUE informational constraints can be created for multiple columns in a table, but only one PRIMARY KEY constraint can be created in a table.
- Multiple informational constraints can be established in a column of a table (because the function that establishing a column or multiple constraints in a column is the same.) Therefore, you are not advised to set up multiple informational constraints in a column, and only one Primary Key type can be set up.
- Table-level COMMENT is not supported.
- Multi-column combination constraints are not supported.
- Different CNs in the same cluster cannot concurrently export data to the same write-only ORC foreign table.
- The catalog of a write-only foreign table in ORC format can only be used as the export catalog of a single foreign table of DWS. It cannot be used for multiple foreign tables, and other components cannot write other files to this catalog.

Example: Creating an HDFS Foreign Table

In HDFS, import the TPC-H benchmark test tables **part** and **region** using Hive. The path of the **part** table is **/user/hive/warehouse/partition.db/part_4**, and that of the **region** table is **/user/hive/warehouse/mppdb.db/region_orc11_64stripe/**.

1. Establish HDFS_Server, with HDFS_FDW or DFS_FDW as the foreign data wrapper.

```
CREATE SERVER hdfs_server FOREIGN DATA WRAPPER HDFS_FDW OPTIONS
(address '10.10.0.100:25000,10.10.0.101:25000',
hdfsconfigpath '/opt/hadoop_client/HDFS/hadoop/etc/hadoop',
type'HDFS');
```

NOTE

- The IP addresses and port numbers of HDFS NameNodes are specified in **OPTIONS**. For details about the port number, search for **dfs.namenode.rpc.port** in the MRS-HDFS service configuration. In this example the port number is 25000.
 - **10.10.0.100:25000,10.10.0.101:25000** indicates the IP addresses and port numbers of the primary and standby HDFS NameNodes. It is the recommended format. Two groups of parameters are separated by commas (,).
2. Create an HDFS foreign table. The HDFS server associated with the table is **hdfs_server**, the corresponding file format of the **ft_region** table on the HDFS server is **'orc'**, and the file directory in the HDFS file system is **'/user/hive/warehouse/mppdb.db/region_orc11_64stripe/'**.

- Create an HDFS foreign table without partition keys.

```
CREATE FOREIGN TABLE ft_region
(
    R_REGIONKEY INT4,
    R_NAME TEXT,
    R_COMMENT TEXT
)
SERVER
    hdfs_server
OPTIONS
(
    FORMAT 'orc',
    encoding 'utf8',
    FOLDERNAME '/user/hive/warehouse/mppdb.db/region_orc11_64stripe/'
)
DISTRIBUTE BY
    roundrobin;
```

- Create an HDFS foreign table with partition keys.

```
CREATE FOREIGN TABLE ft_part
(
    p_partkey int,
    p_name text,
    p_mfgr text,
    p_brand text,
    p_type text,
    p_size int,
    p_container text,
    p_retailprice float8,
    p_comment text
)
SERVER
    hdfs_server
OPTIONS
(
    FORMAT 'orc',
    encoding 'utf8',
    FOLDERNAME '/user/hive/warehouse/partition.db/part_4'
)
DISTRIBUTE BY
```



```
roundrobin
PARTITION BY
(p_mfgr) AUTOMAPPED;
```

NOTE

DWS allows you to specify files using the keyword **filenames** or **foldername**. The latter is recommended. The key word **distribute** specifies the storage distribution mode of the **ft_region** table.

3. View the created server and foreign table.

```
SELECT * FROM pg_foreign_table WHERE ftrelid='ft_region'::regclass;
ftrelid | ftserver | ftwriteonly |          ftoptions
-----+-----+-----+-----
16510 | 16509 | f | {format=orc,foldername=/user/hive/warehouse/mppdb.db/
region_orc11_64stripe/}
(1 row)

select * from pg_foreign_table where ftrelid='ft_part'::regclass;
ftrelid | ftserver | ftwriteonly |          ftoptions
-----+-----+-----+-----
16513 | 16509 | f | {format=orc,foldername=/user/hive/warehouse/partition.db/part_4}
(1 row)
```

Example: Creating an HDFS Write-Only Foreign Table

Export data from the TPC-H benchmark test table **region** to the **/user/hive/warehouse/mppdb.db/region_orc/** directory of the HDFS file system through the HDFS write-only foreign table.

1. Create an HDFS foreign table. The corresponding foreign data wrapper is **HDFS_FDW** or **DFS_FDW**, which is the same as that in [Example 1](#).
2. Create a write-only HDFS foreign table.

```
CREATE FOREIGN TABLE ft_wo_region
(
  R_REGIONKEY INT4,
  R_NAME TEXT,
  R_COMMENT TEXT
)
SERVER
  hdfs_server
OPTIONS
(
  FORMAT 'orc',
  encoding 'utf8',
  FOLDERNAME '/user/hive/warehouse/mppdb.db/region_orc/'
)
WRITE ONLY;
```

3. Writes data to the HDFS file system through a write-only foreign table.
- ```
INSERT INTO ft_wo_region SELECT * FROM region;
```

**Example: Creating an HDFS Write-Only Partitioned Foreign Table (Supported Only by Storage-Compute Decoupling 3.0 Version)**

Export data from the TPC-H benchmark test table **region** table to the **/user/hive/warehouse/mppdb.db/region\_orc/** directory of the HDFS file system through the write-only, multi-level partitioned HDFS foreign table.

1. Create an HDFS foreign table. The corresponding foreign data wrapper is **HDFS\_FDW** or **DFS\_FDW**, which is the same as that in [Example: Creating an HDFS Foreign Table](#).

2. Create a write-only foreign table for HDFS multi-level partitions.

```
CREATE FOREIGN TABLE ft_wo_region_partition
(
 R_REGIONKEY INT4,
 R_NAME TEXT,
 R_COMMENT TEXT
)
SERVER
 hdfs_server
OPTIONS
(
 FORMAT 'orc',
 encoding 'utf8',
 FOLDERNAME '/user/hive/warehouse/mppdb.db/region_orc/'
)
WRITE ONLY
PARTITION BY (R_REGIONKEY, R_NAME);
```

3. Writes data to the HDFS file system through a write-only foreign table.  
INSERT INTO ft\_wo\_region\_partition SELECT \* FROM region;

## Example: Creating an OBS Read-Only Foreign Table (Supported Only by Storage-Compute Decoupling 3.0 version)

Read data from the OBS file system through the foreign table **partsupp**.

1. Create an OBS server.

### NOTICE

- Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.
- For details about the syntax for creating a server, see [CREATE SERVER](#).

```
CREATE SERVER obs_srv FOREIGN DATA WRAPPER dfs_fdw OPTIONS (
 address 'obs.example.com',
 type 'obs',
 access_key 'xxx',
 secret_access_key 'xxx'
);
```

2. Create an OBS foreign table **partsupp** and use the cold cache in the disk cache.

```
CREATE FOREIGN TABLE PARTSUPP
(
 PS_PARTKEY BIGINT NOT NULL,
 PS_SUPPKEY BIGINT NOT NULL,
 PS_AVAILQTY BIGINT NOT NULL,
 PS_SUPPLYCOST DECIMAL(15,2) NOT NULL,
 PS_COMMENT VARCHAR(199) NOT NULL
) SERVER obs_srv options (
 encoding 'utf-8',
 format 'parquet',
 foldername 'xxx',
 cache_policy 'NONE'
) READ ONLY DISTRIBUTE BY ROUNDROBIN;
```

3. Read data from OBS through the foreign table **partsupp**.  
SELECT \* FROM PARTSUPP;

## Example: Creating an HDFS Foreign Table Containing an Informational Constraint (NOT ENFORCED configured)

Perform operations on an HDFS foreign table that includes informational constraints.

- Create an HDFS foreign table with informational constraints.

```
CREATE FOREIGN TABLE ft_region (
 R_REGIONKEY int,
 R_NAME TEXT,
 R_COMMENT TEXT
 , primary key (R_REGIONKEY) not enforced)
SERVER hdfs_server
OPTIONS(format 'orc',
 encoding 'utf8',
 foldername '/user/hive/warehouse/mppdb.db/region_orc11_64stripe')
DISTRIBUTE BY roundrobin;
```

- Check whether the region table has an informational constraint index:

```
SELECT relname,relhasindex FROM pg_class WHERE oid='ft_region'::regclass;
```

| relname   | relhasindex |
|-----------|-------------|
| ft_region | f           |

(1 row)

```
SELECT conname, contype, consoft, conopt, conindid, conkey FROM pg_constraint WHERE conname
='ft_region_pkey';
```

| conname        | contype | consoft | conopt | conindid | conkey |
|----------------|---------|---------|--------|----------|--------|
| ft_region_pkey | p       | t       | t      | 0        | {1}    |

(1 row)

- Delete the informational constraint:

```
ALTER FOREIGN TABLE ft_region DROP CONSTRAINT ft_region_pkey RESTRICT;
```

```
SELECT conname, contype, consoft, conindid, conkey FROM pg_constraint WHERE conname
='ft_region_pkey';
```

| conname | contype | consoft | conindid | conkey |
|---------|---------|---------|----------|--------|
|---------|---------|---------|----------|--------|

(0 rows)

- Add a unique informational constraint for the foreign table:

```
ALTER FOREIGN TABLE ft_region ADD CONSTRAINT constr_unique UNIQUE(R_REGIONKEY) NOT
ENFORCED;
```

Delete the unique informational constraint:

```
ALTER FOREIGN TABLE ft_region DROP CONSTRAINT constr_unique RESTRICT;
```

```
SELECT conname, contype, consoft, conindid, conkey FROM pg_constraint WHERE conname
='constr_unique';
```

| conname | contype | consoft | conindid | conkey |
|---------|---------|---------|----------|--------|
|---------|---------|---------|----------|--------|

(0 rows)

- Add a unique informational constraint for the foreign table:

```
ALTER FOREIGN TABLE ft_region ADD CONSTRAINT constr_unique UNIQUE(R_REGIONKEY) NOT
ENFORCED disable query optimization;
```

```
SELECT relname,relhasindex FROM pg_class WHERE oid='ft_region'::regclass;
```

| relname   | relhasindex |
|-----------|-------------|
| ft_region | f           |

(1 row)

Delete the unique informational constraint:

```
ALTER FOREIGN TABLE ft_region DROP CONSTRAINT constr_unique CASCADE;
```

## Example: Reading OBS JSON Data Using an OBS Foreign Table

Read json data stored in OBS using a foreign table.

1. The following JSON files are on OBS. The JSON objects contain nesting and arrays. Some objects have lost columns, and some object names are duplicate.

```
{"A": "simple1", "B": {"C": "nesting1"}, "D": ["array", 2, {"E": "complicated"}]}
{"A": "simple2", "D": ["array", 2, {"E": "complicated"}]}
{"A": "simple3", "B": {"C": "nesting3"}, "D": ["array", 2, {"E": "complicated3"}]}
{"B": {"C": "nesting4"}, "A": "simple4", "D": ["array", 2, {"E": "complicated4"}]}
{"A": "simple5", "B": {"C": "nesting5"}, "D": ["array", 2, {"E": "complicated5"}]}
```

2. Create **obs\_server**, with **DFS\_FDW** as the foreign data wrapper.

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER DFS_FDW OPTIONS (
 ADDRESS 'obs.example.com',
 ACCESS_KEY 'xxxxxxxxx',
 SECRET_ACCESS_KEY 'yyyyyyyyyyyyyy',
 TYPE 'OBS'
);
```

### NOTE

- **ADDRESS** is the endpoint of OBS. Replace it with the actual endpoint. You can find the domain name by searching for the value of **regionCode** in the **region\_map** file.
  - **ACCESS\_KEY** and **SECRET\_ACCESS\_KEY** are access keys for the cloud account system. Replace the values as needed.
  - **TYPE** indicates the server type. Retain the value **OBS**.
3. Create the OBS foreign table **json\_f** and define the column names. For example, **d#2\_e** indicates that the column is object **e** nested in the 2nd element of array **d**. The OBS server associated with the table is **obs\_server**. **foldername** indicates the data source directory of the foreign table, that is, the OBS directory.

### NOTICE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
CREATE FOREIGN TABLE json_f (
 a VARCHAR(10),
 b_c TEXT,
 d#1 INTEGER,
 d#2_e VARCHAR(30)
)SERVER obs_server OPTIONS (
 foldername '/xxx/xxx/',
 format 'json',
 encoding 'utf8',
 force_mapping 'true'
)distribute by roundrobin;
```

4. Query the foreign table **json\_f**. The fault tolerance parameter **force\_mapping** is enabled by default. If a column is missing in a JSON object, NULL is filled in. If a JSON object name is duplicate, the last name prevails.

```
SELECT * FROM json_f;
 a | b_c | d#1 | d#2_e
-----+-----+-----+-----
simple1 | nesting1 | 2 | complicated1
simple2 | | 2 | complicated2
simple3 | nesting3 | 2 | complicated3
```

```
simple4 | nesting4 | 2 | complicated4
repeat | nesting5 | 2 | complicated5
(5 rows)
```

## Example: Accessing Data on DLI Through a DLI Foreign Table

Read a DLI multi-version foreign table using a foreign table. Only DLI 8.1.1 and later support the multi-version foreign table example.

1. Create **dli\_server**, with **DFS\_FDW** as the foreign data wrapper.

```
CREATE SERVER dli_server FOREIGN DATA WRAPPER DFS_FDW OPTIONS (
 ADDRESS 'obs.example.com',
 ACCESS_KEY 'xxxxxxxxx',
 SECRET_ACCESS_KEY 'yyyyyyyyyyyyyy',
 TYPE 'DLI',
 DLI_ADDRESS 'dli.example.com',
 DLI_ACCESS_KEY 'xxxxxxxxx',
 DLI_SECRET_ACCESS_KEY 'yyyyyyyyyyyyyy'
);
```

### NOTE

- **ADDRESS** is the endpoint of OBS. **DLI\_ADDRESS** is the endpoint of DLI. Replace it with the actual endpoint.
  - **ACCESS\_KEY** and **SECRET\_ACCESS\_KEY** are access keys for the cloud account system to access OBS. Use the actual value.
  - **DLI\_ACCESS\_KEY** and **DLI\_SECRET\_ACCESS\_KEY** are access keys for the cloud account system to access DLI. Use the actual value.
  - **TYPE** indicates the server type. Retain the value **DLI**.
2. Create the OBS foreign table **customer\_address** for accessing DLI. The table does not contain partition columns, and the DLI server associated with the table is **dli\_server**. Where, the **project\_id** is **xxxxxxxxxxxxxxxxxx**, the **database\_name** on DLI is **database123**, and the **table\_name** of the table to be accessed is **table456**. Replace them based on the actual requirements.

### NOTICE

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
CREATE FOREIGN TABLE customer_address
(
 ca_address_sk integer not null,
 ca_address_id char(16) not null,
 ca_street_number char(10) ,
 ca_street_name varchar(60) ,
 ca_street_type char(15) ,
 ca_suite_number char(10) ,
 ca_city varchar(60) ,
 ca_county varchar(30) ,
 ca_state char(2) ,
 ca_zip char(10) ,
 ca_country varchar(20) ,
 ca_gmt_offset decimal(36,33) ,
 ca_location_type char(20) ,
)
SERVER dli_server OPTIONS (
 FORMAT 'ORC',
 ENCODING 'utf8',
```

```
DLI_PROJECT_ID 'xxxxxxxxxxxxxx',
DLI_DATABASE_NAME 'database123',
DLI_TABLE_NAME 'table456'
)
DISTRIBUTE BY roundrobin;
```

3. Query data in a DLI multi-version table using a foreign table.

```
SELECT COUNT(*) FROM customer_address;
count

20
(1 row)
```

## Helpful Links

- For details about the syntax for modifying and deleting a foreign table, see [ALTER FOREIGN TABLE \(for HDFS or OBS\)](#) and [DROP FOREIGN TABLE](#).

### 21.13.4.9 CREATE FOREIGN TABLE (for OBS Import and Export)

#### Function

**CREATE FOREIGN TABLE** creates an OBS foreign table in the current database for parallel data import and export of OBS data. You do not need to create an external server. The **gsmpp\_server** created by the database by default can be used.

This syntax supports only TEXT and CSV data on OBS buckets. If you need to process ORC, CARBONDATA, and PARQUET data, see [CREATE FOREIGN TABLE \(SQL on OBS or Hadoop\)](#) and create an external server.

A foreign table is a virtual table within a database or big data platform that does not physically store data. Instead, it serves as a mapping to data residing in external storage systems, such as HDFS or OBS. This mapping is established using metadata, which includes the table structure, the location of the data, and its file format. It enables users to directly access or operate external data using standard SQL statements. A foreign table serves as a data proxy that logically maps data in external file systems to table structures that can be identified by databases.

#### Precautions

- Only the data in text and CSV formats is supported, and the OBS connection should be configured.
- When using an OBS foreign table to access data in an OBS bucket, you need to ensure that your DWS cluster and OBS bucket are in the same region.
- Foreign tables are classified into read-only foreign tables (**READ ONLY**) and write-only foreign tables (**WRITE ONLY**). By default, foreign tables are read-only. When importing OBS data, you need to set the foreign table to **READ ONLY**. When exporting data to OBS, you need to set the foreign table to **WRITE ONLY**.
- Only the system administrator **dbadmin** or a common user who has been granted the USEFT permission can perform foreign table operations.  

```
ALTER USER user_name USEFT;
```
- The distribution mode of an OBS foreign table does not need to be explicitly specified. The default mode is **ROUNDROBIN**.

- Only constraints in **Informational Constraints** take effect for the created foreign table.
- OBS buckets cannot contain Chinese paths.

**Table 21-178** Read and write formats supported by OBS foreign tables

| Data Type  | DIST_FDW  |            |
|------------|-----------|------------|
| -          | READ ONLY | WRITE ONLY |
| ORC        | ×         | ×          |
| PARQUET    | ×         | ×          |
| CARBONDATA | ×         | ×          |
| TEXT       | √         | √          |
| CSV        | √         | √          |
| JSON       | ×         | ×          |

## Syntax

```
CREATE FOREIGN TABLE [IF NOT EXISTS] table_name
({ column_name type_name [column_constraint]
 | LIKE source_table | table_constraint [, ...]} [, ...])
SERVER server_name
OPTIONS ({ option_name ' value ' } [, ...])
[{ WRITE ONLY | READ ONLY }]
[WITH error_table_name | LOG INTO error_table_name]
[PER NODE REJECT LIMIT 'value'] ;
```

- **column\_constraint** is as follows:  
[CONSTRAINT constraint\_name]  
{PRIMARY KEY | UNIQUE}  
[NOT ENFORCED [ENABLE QUERY OPTIMIZATION | DISABLE QUERY OPTIMIZATION] | ENFORCED]
- **table\_constraint** is as follows:  
[CONSTRAINT constraint\_name]  
{PRIMARY KEY | UNIQUE} (column\_name)  
[NOT ENFORCED [ENABLE QUERY OPTIMIZATION | DISABLE QUERY OPTIMIZATION] | ENFORCED]

## Parameter Description

**Table 21-179** CREATE FOREIGN TABLE (for OBS import and export) parameters

| Parameter     | Description                                                                        | Value Range or Example |
|---------------|------------------------------------------------------------------------------------|------------------------|
| IF NOT EXISTS | Sends a notice, but does not throw an error, if a table with the same name exists. | -                      |

| Parameter          | Description                                                                                                                                                                                                    | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| table_name         | Specifies the name of the foreign table.                                                                                                                                                                       | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                                                                                                                                         |
| column_name        | Specifies the name of a column in the foreign table.                                                                                                                                                           | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                                                                                                                                         |
| type_name          | Specifies the data type of the column.                                                                                                                                                                         | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| SERVER server_name | Specifies the server name of the foreign table. For OBS foreign tables used for data import and export, you can use <b>gsmpp_server</b> created by the initial database by default or use a customized server. | <ul style="list-style-type: none"><li>• If a custom server is used, the foreign data wrapper should be <b>dist_fdw</b>.</li><li>• For clusters of 8.2.0 and later versions, you can specify the following OBS access parameters in the customized <b>dist_fdw</b> server: <b>access_key</b>, <b>secret_access_key</b>, and <b>security_token</b>. If the preceding parameters are specified in the server, you do not need to specify them again in the foreign table.</li></ul> |
| OPTIONS            | Specifies parameters of foreign table data.                                                                                                                                                                    | <ul style="list-style-type: none"><li>• Data format parameters. For details, see <a href="#">Table 21-180</a>.</li><li>• Error-tolerance parameters. For details, see <a href="#">Table 21-181</a>.</li><li>• Performance parameters. For details, see <a href="#">Table 21-182</a>.</li></ul>                                                                                                                                                                                   |
| READ ONLY          | Specifies whether a foreign table is read-only. This parameter is available only for data import.                                                                                                              | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |



| Parameter                        | Description                                                                                                                                                                                                                                | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WRITE ONLY                       | Specifies whether a foreign table is write-only. This parameter is available only for data export.                                                                                                                                         | -                                                                                                                                                                                                                                                                                                                                                                                                                          |
| WITH<br>error_table_name         | Data format errors during import are recorded in the table specified by <b>error_table_name</b> . You can query this table after the import to obtain error details. This parameter is available only after <b>reject_limit</b> is set.    | A string, which must comply with the naming convention.<br><b>CAUTION</b><br>To be compatible with PostgreSQL open source interfaces, you are advised to replace this syntax with <b>LOG INTO</b> . When this parameter is specified, an error table is automatically created.                                                                                                                                             |
| LOG INTO<br>error_table_name     | Data format errors during import are recorded in the table specified by <b>error_table_name</b> . You can query this table after the import to obtain error details.                                                                       | A string, which must comply with the naming convention. <ul style="list-style-type: none"><li>• This parameter is available only after <b>PER NODE REJECT LIMIT</b> is set.</li><li>• When this parameter is specified, an error table is automatically created.</li></ul>                                                                                                                                                 |
| PER NODE REJECT<br>LIMIT 'value' | This parameter specifies the allowed number of data format errors on each DN during data import. If the number of errors exceeds the specified value on any DN, data import fails, an error is reported, and the system exits data import. | An <b>unlimited</b> integer. If this parameter is not specified, an error message is returned immediately.<br><b>CAUTION</b><br>This syntax specifies the error tolerance of a single node. Examples of data format errors include the following: a column is lost, an extra column exists, a data type is incorrect, and encoding is incorrect. When a non-data format error occurs, the whole data import process stops. |

| Parameter    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Value Range or Example                                                                                                                                                                     |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NOT ENFORCED | <p>Specifies that the created constraint is an informational constraint, which is not forcibly verified by the database. This option is used with <b>ENABLE QUERY OPTIMIZATION</b>.</p> <ul style="list-style-type: none"><li>• Informational rather than mandatory: You still need to specify a unique constraint for a column using <b>PRIMARY KEY</b> or <b>UNIQUE</b> to optimize the query execution plan.</li><li>• No mandatory verification: The database does not check whether the data in the external data source actually meets constraint conditions (for example, whether duplicate values exist).</li><li>• Performance optimization: The query optimizer is allowed to use an informational constraint to generate a more efficient execution plan. However, you need to ensure that the data meets constraint conditions, or the query result may be incorrect.</li><li>• <b>Risk</b>: If the actual data violates constraint conditions (for example, duplicate values exist), the query result may be incorrect (for example, the aggregation result is</li></ul> | <p>For example, the primary key is declared, but the verification is not mandatory.</p> <pre>CREATE FOREIGN TABLE hdfs_users<br/>( user_id INT PRIMARY KEY NOT<br/>ENFORCED,<br/>...</pre> |

| Parameter                     | Description                                                                                                                                                                               | Value Range or Example                                                                                                                                                                                                                                                                                                       |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                               | abnormal or redundant records are generated during JOIN operations).<br>For more information, see <a href="#">Informational Constraints</a> .                                             |                                                                                                                                                                                                                                                                                                                              |
| ENFORCED                      | Specifies that the created constraint is an enforced constraint that is forcibly validated by the database. This parameter is reserved. Currently, DWS does not support <b>ENFORCED</b> . | -                                                                                                                                                                                                                                                                                                                            |
| PRIMARY KEY<br>(column_name)  | Specifies the informational constraint on <b>column_name</b> .                                                                                                                            | String, which must conform to the identifier naming conventions. The column name must exist.                                                                                                                                                                                                                                 |
| ENABLE QUERY<br>OPTIMIZATION  | Enables the query optimizer to use informational constraints to generate a more efficient execution plan. This parameter is used together with <b>NOT ENFORCED</b> .                      | Create a foreign table, add a unique constraint to the <b>p_partkey int</b> column, do not enforce the constraint, and allow the optimizer to use the informational constraint to generate a better execution plan.<br>CREATE FOREIGN TABLE ft_part (<br>p_partkey int UNIQUE NOT ENFORCED ENABLE QUERY OPTIMIZATION,<br>... |
| DISABLE QUERY<br>OPTIMIZATION | Disables the query optimizer to use informational constraints to generate an efficient execution plan.                                                                                    | -                                                                                                                                                                                                                                                                                                                            |

**Table 21-180** OPTIONS parameters of the data format

| Parameter         | Description                                                                                                                                                                                                                                                                                                             | Value Range or Example |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| encrypt           | Specifies whether HTTPS is enabled for data transfer. <b>on</b> enables HTTPS and <b>off</b> disables it (in this case, HTTP is used). The default value is <b>off</b> .                                                                                                                                                | -                      |
| access_key        | Indicates the access key (AK, obtained from the user information on the console) used for the OBS access protocol. When you create a foreign table, its AK value is not encrypted and saved to the metadata table of the database. The correctness of the parameter is not verified when a foreign table is created.    | -                      |
| secret_access_key | Indicates the secret access key (SK, obtained from the user information on the console) used for the OBS access protocol. When you create a foreign table, its SK value is encrypted and saved to the metadata table of the database. The correctness of the parameter is not verified when a foreign table is created. | -                      |

| Parameter      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Value Range or Example                                                                                                                                         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| security_token | <p>Corresponds to the <b>SecurityToken</b> value of the temporary security credential in IAM. A temporary AK, a temporary SK, and a temporary security token form a temporary security credential. The temporary security credential is valid for no more than 24 hours. This parameter is supported by version 8.2.0 or later clusters.</p> <p><b>CAUTION</b></p> <ul style="list-style-type: none"><li>This parameter is supported by version 8.2.0 or later clusters.</li><li>When this parameter is used, <b>access_key</b> and <b>secret_access_key</b> correspond to the temporary AK and SK, respectively.</li></ul> | -                                                                                                                                                              |
| chunksize      | <p>Specifies the cache read by each OBS thread on a DN. Its value range is 8 to 512 in the unit of MB. Its default value is <b>64</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | -                                                                                                                                                              |
| location       | <p>Specifies the directory where OBS data is stored. The value can be described as a URL in the format of <b>obs://OBS bucket name/folder name</b>. Multiple URLs are separated by vertical bars ( ). Ensure that the OBS bucket and DWS cluster are in the same region. Cross-region access to OBS bucket data is not supported.</p> <p>For details about how to use this parameter, see <a href="#">Location Parameter Description</a>.</p>                                                                                                                                                                               | <p>Example:</p> <pre>OPTIONS<br/>( encoding 'utf8',      location<br/>'obs://&lt;obs_bucket_name&gt;/traffic-<br/>data/gcxx',<br/>format 'text',<br/>...</pre> |

| Parameter | Description                                                                                                                                                                                                                                                                                           | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| region    | <p>(Optional) specifies the value of <b>regionCode</b>, region information on the cloud.</p> <p>If the <b>region</b> parameter is explicitly specified, the value of <b>region</b> will be read. If the <b>region</b> parameter is not specified, the value of <b>defaultRegion</b> will be read.</p> | <p>Note the following when setting parameters for importing or exporting OBS foreign tables in TEXT or CSV format:</p> <ul style="list-style-type: none"><li>• The <b>location</b> parameter is mandatory. The prefixes <b>gsobs</b> and <b>obs</b> indicate file locations on OBS. The <b>gsobs</b> prefix should be followed by <i>obs url</i>, <i>bucket</i>, and <i>prefix</i>. The <b>obs</b> prefix should be followed by <i>bucket</i> or <i>prefix</i>.</li><li>• The data sources of multiple buckets are separated by vertical bars ( ), for example, <b>LOCATION 'obs://bucket1/folder/   obs://bucket2/'</b>. The database scans all objects in the specified folders.</li></ul> |

| Parameter | Description                                                                           | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------|---------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| format    | Specifies the format of the source data file in a foreign table.                      | <p>CSV or TEXT (default value). DWS only supports CSV and TEXT formats.</p> <ul style="list-style-type: none"><li>• CSV (comma-separated format):<ul style="list-style-type: none"><li>– The CSV file can process linefeeds efficiently, but cannot process certain special characters very well.</li><li>– A CSV file is composed of records that are separated as columns by delimiters. Each record shares the same column sequence.</li></ul></li><li>• TEXT (text format):<p>Records are separated as columns by linefeed. The TEXT file can process special characters efficiently, but cannot process linefeeds well.</p></li></ul> |
| header    | Specifies whether a file contains a header with the names of each column in the file. | <p>When OBS exports data, this parameter cannot be set to <b>true</b>. Use the default value <b>false</b>, indicating that the first row of the exported data file is not the header.</p> <p>When data is imported, if <b>header</b> is <b>on</b>, the first row of the data file will be identified as title row and ignored. If <b>header</b> is <b>off</b>, the first row will be identified as a data row.</p> <p>Valid value: <b>true</b>, <b>on</b>, <b>false</b>, and <b>off</b>. The default value is <b>false</b> or <b>off</b>.</p>                                                                                              |

| Parameter | Description                                                                                                                                                     | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| delimiter | Specifies the column delimiter of data, and uses the default delimiter if it is not set. The default delimiter of TEXT is a tab and that of CSV is a comma (,). | <p>The value of <b>delimiter</b> can be a multi-character delimiter whose length is less than or equal to 10 bytes.</p> <ul style="list-style-type: none"><li>• The delimiter of TEXT cannot be <b>\r</b> or <b>\n</b>.</li><li>• A delimiter cannot be the same as the <b>null</b> value. The delimiter for the CSV format cannot be same as the <b>quote</b> value.</li><li>• The separator of TEXT data cannot contain letters, digits, backslashes (\), and periods (.).</li><li>• The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.</li><li>• You are advised to use a multi-character string, such as the combination of the dollar sign (\$), caret (^), and ampersand (&amp;), or invisible characters, such as 0x07, 0x08, and 0x1b as the delimiter.</li></ul> |
| quote     | Specifies the quotation mark for the CSV format. The default value is a double quotation mark (").                                                              | <ul style="list-style-type: none"><li>• The <b>quote</b> value cannot be the same as the delimiter or <b>null</b> value.</li><li>• The <b>quote</b> value must be a single-byte character.</li><li>• Invisible characters are recommended as <b>quote</b> values, such as 0x07, 0x08, and 0x1b.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |



| Parameter  | Description                                                                                    | Value Range or Example                                                                                                                                                                                                                                                                                                                                               |
|------------|------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| escape     | Specifies an escape character for a CSV file. The value must be a single-byte character.       | The default value is a double quotation mark ("). If the value is the same as the <b>quote</b> value, it will be replaced with \0.                                                                                                                                                                                                                                   |
| null       | Specifies how to represent a null value.                                                       | <ul style="list-style-type: none"><li>• The default value is \N for the TEXT format.</li><li>• The default value for the CSV format is an empty string without quotation marks.</li><li>• The <b>null</b> value cannot be \r or \n. The maximum length is 100 characters.</li><li>• The null value cannot be the same as the delimiter or quote parameter.</li></ul> |
| noescaping | Specifies whether to escape the backslash (\) and its following characters in the TEXT format. | <b>noescaping</b> is available only for the TEXT format. <b>true/on</b> or <b>false/off</b> . The default value is <b>false</b> or <b>off</b> .                                                                                                                                                                                                                      |

| Parameter | Description                                                                                                                                                                                         | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| encoding  | Specifies the encoding of a data file, that is, the encoding used to parse, check, and generate a data file. Its default value is the default <b>client_encoding</b> value of the current database. | <p>Before you import foreign tables, it is recommended that you set <b>client_encoding</b> to the file encoding format, or a format matching the character set of the file. Otherwise, unnecessary parsing and check errors may occur, leading to import errors, rollback, or even invalid data import. Before exporting foreign tables, you are also advised to specify this parameter, because the export result using the default character set may not be what you expect.</p> <p>If this parameter is not specified when you create a foreign table, a warning message will be displayed on the client.</p> <p><b>CAUTION</b></p> <ul style="list-style-type: none"><li>Currently, OBS cannot parse a file using multiple character sets during foreign table import.</li><li>Currently, OBS cannot write a file using multiple character sets during foreign table export.</li></ul> |

| Parameter            | Description                                                                                                     | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|-----------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| eol                  | Specifies the newline character style of the imported or exported data file.                                    | <p>Multi-character newline characters are supported, but the newline character cannot exceed 10 bytes. Common newline characters include <code>\r</code> (0x0D), <code>\n</code> (0x0A), and <code>\r\n</code> (0x0D0A). Special newline characters include <code>\$</code> and <code>#</code>.</p> <ul style="list-style-type: none"><li>• The <b>eol</b> parameter supports only the TEXT format for data import and export.</li><li>• The value of the <b>eol</b> parameter cannot be the same as that of <b>DELIMITER</b> or <b>NULL</b>.</li><li>• The value of the <b>eol</b> parameter cannot contain digits, letters, or periods (.).</li></ul> |
| date_format          | Specifies the DATE format for data import. This syntax is available only for READ ONLY foreign tables.          | <p>Valid DATE formats. For details, see <a href="#">Date and Time Processing Functions and Operators</a>.</p> <p><b>NOTE</b><br/>If ORACLE is specified as the compatible database, the DATE format is TIMESTAMP. For details, see <b>timestamp_format</b> below.</p>                                                                                                                                                                                                                                                                                                                                                                                   |
| time_format          | Specifies the TIME format for data import. This syntax is available only for READ ONLY foreign tables.          | Valid TIME formats. Time zones are not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| timestamp_format     | Specifies the TIMESTAMP format for data import. This syntax is available only for READ ONLY foreign tables.     | Valid TIMESTAMP formats. Time zones are not supported.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| smalldatetime_format | Specifies the SMALLDATETIME format for data import. This syntax is available only for READ ONLY foreign tables. | Valid SMALLDATETIME formats.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

| Parameter | Description                                         | Value Range or Example                                                                                                                                                                           |
|-----------|-----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| bom       | Indicates whether a CSV file contains the utf8 BOM. | This parameter is valid only when the foreign table is read-only and uses UTF-8 code.<br><br>Value range: <b>true</b> , <b>on</b> , <b>false</b> , and <b>off</b><br>Default value: <b>false</b> |

**Table 21-181** OPTIONS fault tolerance parameters

| Parameter           | Description                                                                                                                                                                                 | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fill_missing_fields | Specifies how to handle the problem that the last column of a row in the source file is lost during data import.                                                                            | <b>true/on</b> or <b>false/off</b> . The default value is <b>false</b> or <b>off</b> . <ul style="list-style-type: none"><li>If this parameter is set to <b>true</b> or <b>on</b> and the last column of a data row in a source data file is lost, the column will be replaced with <b>NULL</b> and no error message will be generated.</li><li>If this parameter is set to <b>false</b> or <b>off</b> and the last column of a data row in a source data file is lost, the following error information will be displayed:<br/>missing data for column "tt"</li></ul>                                                                                                                                       |
| ignore_extra_data   | Specifies whether to ignore excessive columns when the number of columns in a source data file exceeds that defined in the foreign table. This parameter is available only for data import. | <b>true/on</b> or <b>false/off</b> . The default value is <b>false</b> or <b>off</b> . <ul style="list-style-type: none"><li>If this parameter is set to <b>true</b> or <b>on</b> and the number of source data files exceeds the number of foreign table columns, excessive columns will be ignored.</li><li>If this parameter is set to <b>false</b> or <b>off</b> and the number of source data files exceeds the number of foreign table columns, the following error information will be displayed:<br/>extra data after last expected column</li></ul> <b>CAUTION</b><br>If the linefeed at the end of a row is lost and this parameter is set to <b>true</b> , data in the next row will be ignored. |

| Parameter      | Description                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| reject_limit   | Specifies the maximum number of data format errors allowed during the data import. If the number of data format errors does not reach the maximum, the data import is successful. | <p>The value is an integer or <b>unlimited</b>. If this parameter is not specified, an error message is returned immediately.</p> <p><b>CAUTION</b></p> <p>You are advised to replace this syntax with <b>PER NODE REJECT LIMIT 'value'</b>.</p> <p>Examples of data format errors include the following: a column is lost, an extra column exists, a data type is incorrect, and encoding is incorrect. Once a non-data format error occurs, the whole data import process is stopped.</p> |
| force_save_err | Indicates whether to save the error information to the error table after the import exits due to an error.                                                                        | <p><b>true/on</b> or <b>false/off</b>. The default value is <b>false</b> or <b>off</b>.</p> <p>This parameter is used together with <b>reject_limit</b>. Once this parameter is enabled:</p> <ul style="list-style-type: none"><li>• If <b>reject_limit</b> is not specified, an error record will be retained in the error table.</li><li>• If <b>reject_limit</b> is set to <i>N</i>, <i>N+1</i> error records will be retained in the error table.</li></ul>                             |

| Parameter     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| obs_null_file | <p>Imports and exports empty files between DWS and OBS.</p> <p><b>CAUTION</b></p> <ul style="list-style-type: none"><li>• This parameter is supported only in 8.2.1 or later.</li><li>• If <b>obs_null_file</b> is set to <b>true</b> or <b>on</b> and the export directory contains only the <b>_SUCCESS</b> empty file, the empty table can be exported repeatedly, while if <b>obs_null_file</b> is set to <b>false</b> or <b>off</b>, the empty table cannot be exported repeatedly.</li><li>• If <b>obs_null_file</b> is set to <b>true</b> or <b>on</b> and files are imported from multiple buckets, an error is reported for the first path that the file does not exist.</li></ul> | <p><b>true/on</b> or <b>false/off</b>. The default value is <b>false</b> or <b>off</b>.</p> <p>If <b>obs_null_file</b> is set to <b>true</b> or <b>on</b>:</p> <ul style="list-style-type: none"><li>• When an empty table is exported from DWS, an empty file named <b>_SUCCESS</b> is generated, indicating that the export is successful. When a non-empty table is exported, the original table and an empty file named <b>_SUCCESS</b> is generated.</li><li>• When a file is imported to DWS, if the file does not exist or the path is incorrect, the following error information is displayed:<br/>No such file or directory: 'XXX'</li></ul> |

| Parameter                             | Description                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>compatible_illegal_chars</code> | Specifies whether to enable fault tolerance on invalid characters during data import. This syntax is available only for READ ONLY foreign tables. | <p><b>true/on</b> or <b>false/off</b>. The default value is <b>false</b> or <b>off</b>.</p> <ul style="list-style-type: none"><li>If this parameter is set to <b>true</b> or <b>on</b>, invalid characters are tolerated and imported to the database after conversion.</li><li>If this parameter is set to <b>false</b> or <b>off</b> and an error occurs when there are invalid characters, the import will be interrupted.</li><li>The rule of error tolerance when you import invalid characters is as follows:<ol style="list-style-type: none"><li><code>\0</code> is converted to a space.</li><li>Other invalid characters are converted to question marks.</li><li>Setting <b>compatible_illegal_chars</b> to <b>true</b> or <b>on</b> enables toleration of invalid characters. If <b>NULL</b>, <b>DELIMITER</b>, <b>QUOTE</b>, and <b>ESCAPE</b> are set to spaces or question marks, errors like "illegal chars conversion may confuse COPY escape 0x20" will be displayed to prompt the user to modify parameters that may cause confusion, preventing importing errors.</li></ol></li></ul> <p><b>CAUTION</b><br/>On a Windows platform, if OBS reads data files using the TEXT format, 0x1A will be treated as an EOF symbol and a parsing error will occur. It is the implementation constraint of the Windows platform. Since OBS on a Windows platform does not support BINARY read, the data can be read by OBS on a Linux platform.</p> |

Table 21-182 OPTIONS performance parameters

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file_split_threshold | This parameter is used to optimize the performance of importing data in TEXT format. It specifies the lower limit of the logical block size of a file. If this parameter is specified, large files are split based on the actual file and DN status to improve the import concurrency. The purpose is to evenly distribute tasks on each DN. Therefore, this parameter can be used in scenarios where the number of files is less than the number of DNs or the file size is unbalanced. | <b>0 to 2147483647</b> , in MB The default value is <b>0</b> , indicating that files are not split. <ul style="list-style-type: none"><li>• This parameter is supported only in 8.2.0 or later.</li><li>• This parameter supports only READ ONLY foreign tables in TEXT format.</li><li>• This parameter specifies the lower limit of the logical block size of a file. It does not specify a block size. For example, if the current file size is 1,024 MB and the number of DNs is 4, If the value of <b>file_split_threshold</b> is less than <b>256</b>, the file is evenly divided into four blocks, and a 256 MB file import task is allocated to each DN. When <b>file_split_threshold</b> is set to <b>500</b>, the file is split into 500 MB and 524 MB and allocated to two DNs because the block size cannot be less than 500 MB. This parameter is also applicable to multiple files.</li><li>• Unless there are clear requirements for block sizes, you are advised to set this parameter to a small value, for example, 10. Otherwise, the concurrency may be affected.</li></ul> |

### Location Parameter Description

- The URL of a read-only foreign table (the default permission is read-only) can end with the path prefix or the full path of the target object in the format of **obs://Bucket/Prefix**. *Prefix* indicates the prefix of an object path, for example, **obs://mybucket/tpch/nation/**.
- If the **region** parameter is explicitly specified in **obs://Bucket/Prefix**, the value of **region** will be read. If the **region** parameter is not specified, the value of **defaultRegion** will be read.



- The URL of a writable foreign table does not need to contain a file name. You can specify only one data source location for a foreign table. The directory corresponding to the location must be created before you specify the location.
- URLs specified for a read-only foreign table must be different.
- Specify **location** when inserting data to a foreign table.
- Parameter **LOCATION** supports prefixes **gsobs** and **obs**, which are identified as OBS information. **LOCATION** should be followed by **gsobs**, *OBS URL*, and *Bucket*, or by **obs** and *Bucket*.

When importing and exporting data, you are advised to use the **location** parameter as follows:

- You are advised to specify a file name for **location** during data import. If you only specify an OBS bucket or directory, all text files in it will be imported. An error message will be reported if the data format is incorrect. If you set fault tolerance, a large amount of data may be imported to the fault-tolerant table.
- Multiple files in an OBS bucket can be imported at the same time. The matched files are imported based on the file name prefix.  
For example, you can identify and import the following two files after specifying the prefix **mybucket/input\_data/product\_info** in **location**:  
mybucket/input\_data/product\_info.0  
mybucket/input\_data/product\_info.1
- If you specify a file name, for example, **1.csv**, then other files (like **1.csv1** or **1.csv22**) starting with **1.csv** in the bucket or directory where **1.csv** resides will be automatically imported. That is, **1.csv1** and **1.csv22** are automatically imported.
- To specify multiple URLs in OBS mode, separate URLs by using vertical bars (|). In gsobs mode, only one URL can be specified.
- During data export, a directory is generated for **location** by default. If you specify only a file name, the system automatically creates a directory whose name starts with the file name and then generates the file that stores the exported data. The file name is automatically generated by DWS.
- You can specify one path for **location** only during data export.

## Examples

Create a foreign table named **OBS\_ft** to import data in the .txt format from OBS to the **row\_tbl** table.

---

### CAUTION

// Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

---

```
DROP FOREIGN TABLE IF EXISTS OBS_ft;

CREATE FOREIGN TABLE OBS_ft
(a int, b int)
SERVER gsmpp_server
OPTIONS
(location 'obs://gaussdbcheck/obs_ddl/test_case_data/txt_obs_informatonal_test001',
```

```
format 'text',
encoding 'utf8',
chunksize '32',
encrypt 'on',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
delimiter E'\x08')
read only;

DROP TABLE row_tbl;

CREATE TABLE row_tbl(a int, b int);

INSERT INTO row_tbl select * from OBS_ft;
```

## Helpful Links

[ALTER FOREIGN TABLE \(for HDFS or OBS\), DROP FOREIGN TABLE](#)

## Optimization

- delimiter
  - A delimiter cannot be `\r` or `\n`, or the same as the **null** value. The delimiter of CSV cannot be same as the **quote** value.
  - The data length of a single row should be less than 1 GB. A row that has many columns using long delimiters cannot contain much valid data.
  - You are advised to use a multi-character string, such as the combination of the dollar sign (\$), caret (^), and ampersand (&), or invisible characters, such as 0x07, 0x08, and 0x1b as the delimiter.
- quote
  - The value must be a single-byte character. The **quote** value cannot be the same as the delimiter or **null** value.
  - Invisible characters are recommended as **quote** values, such as 0x07, 0x08, and 0x1b.
- mode Normal
  - Supports all file types (including CSV, TEXT, and FIXED). To import data, you need to enable GDS on the data server.
- mode Shared
  - Supports the TEXT format. It does not require GDS, but all the user data has to be mounted to the same path of all the nodes through NFS.
- mode Private
  - Used in scenarios where user data has been stored under the same path as the local directory of DNs.

### 21.13.4.10 CREATE FOREIGN TABLE (for GDS Import and Export)

#### Function

**CREATE FOREIGN TABLE** creates a GDS foreign table in the current database for concurrent data import and export. The GDS foreign table can be read-only or write-only, used for concurrent data import and export, respectively. The foreign table is read-only by default.

GDS is a command line tool running on the Linux OS. It works with the foreign table mechanism to quickly import and export data. The GDS tool package needs to be installed on the server where the data source file is located. This server is called the data server or the GDS server.

## Precautions

- Only the system administrator **dbadmin** or a common user who has been granted the USEFT permission can perform foreign table operations.  
`ALTER USER user_name USEFT;`
- The distribution mode of a GDS foreign table does not need to be explicitly specified. The default is **ROUNDROBIN**.
- All constraints (including column and row constraints) are invalid to the GDS foreign table.
- GDS supports the following file formats: TEXT, CSV, and FIXED.

## Syntax

```
CREATE FOREIGN TABLE [IF NOT EXISTS] table_name
([{ column_name type_name POSITION(offset,length) | LIKE source_table } [, ...]])
SERVER gsmpp_server
OPTIONS ({ option_name 'value' } [, ...])
[{ WRITE ONLY | READ ONLY }]
[WITH error_table_name | LOG INTO error_table_name]
[REMOTE LOG 'name']
[PER NODE REJECT LIMIT 'value']
[TO { GROUP groupname | NODE (nodename [, ...]) }];
```

## Parameter Description

**Table 21-183** CREATE FOREIGN TABLE (for GDS import and export) parameters

| Parameter     | Description                                                                        | Value Range                                                                                                              |
|---------------|------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| IF NOT EXISTS | Sends a notice, but does not throw an error, if a table with the same name exists. | -                                                                                                                        |
| table_name    | Specifies the name of the foreign table.                                           | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| column_name   | Specifies the name of a column in the foreign table.                               |                                                                                                                          |
| type_name     | Specifies the data type of the column.                                             | -                                                                                                                        |

| Parameter                | Description                                                                                                                                                                                    | Value Range                                                                                                                                                                                                                                                                                    |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| POSITION(offset, length) | Defining the location of each column in the data file in fixed length mode.<br><br><b>offset</b> is the start of the column in the source file, and <b>length</b> is the length of the column. | <b>offset</b> must be greater than 0 bytes, and its unit is byte.<br><br>The length of each record must be less than or equal to 1 GB. By default, columns not in the file are replaced with null.                                                                                             |
| SERVER gsmpp_server      | Specifies the server name of the foreign table. For the GDS foreign table, its server is created by initial database, which is <b>gsmpp_server</b> .                                           | -                                                                                                                                                                                                                                                                                              |
| OPTIONS                  | Specifies parameters of foreign table data.                                                                                                                                                    | <ul style="list-style-type: none"><li>• Data format parameters. For details, see <a href="#">Table 21-184</a>.</li><li>• Error-tolerance parameters. For details, see <a href="#">Table 21-185</a>.</li><li>• Performance parameters. For details, see <a href="#">Table 21-186</a>.</li></ul> |
| READ ONLY                | Specifies whether a foreign table is read-only. This parameter is available only for data import.                                                                                              | -                                                                                                                                                                                                                                                                                              |

**Table 21-184** OPTIONS parameters of the data format

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| location  | <p>Specifies the data source location of the foreign table, which can be expressed through URLs. Separate URLs with vertical bars ( ).</p> <p>Currently, GDS can automatically create a directory defined by a foreign table during data export. If foreign table <b>location</b> specified as <b>gsfs://192.168.0.91:5000/2019/09</b> for an export task, the <b>2019/09</b> subdirectory will be automatically created in the GDS data directory if it does not already exist. Manual creation of the directory specified in the foreign table is not necessary.</p> | <ul style="list-style-type: none"><li>For a read-only foreign table imported by GDS from a remote server in parallel, its URL must end with its corresponding schema or file name. (Read-only is the default file attribute.)<br/>For example: <b>gsfs://192.168.0.90:5000/*</b> or <b>file:///data/data.txt</b> or <b>gsfs://192.168.0.90:5000/*   gsfs://192.168.0.91:5000/*</b>.</li><li>For a writable foreign table used for GDS to export data to a remote server in parallel, file names are not required in URLs. If the data source location is a remote URL, for example, <b>gsfs://192.168.0.90:5000/</b>, multiple data sources can be specified. If the number of exported data file locations is less than or equal to the number of DN, when you use the foreign table for export, data is evenly distributed to each data source location. If the number of exported data file locations is greater than the number of DN, when you export data, the data is evenly distributed to data source locations corresponding to the DN. Blank data files are created on the excess data source locations.</li><li>For a foreign table used for GDS to import data from a remote server in parallel, the number of URLs must be less than the number of DN, and URLs containing the same location cannot be used.</li><li>If the URL begins with <b>gsfs://</b>, data is imported and exported in encryption mode, and DOP cannot exceed 10.</li><li>During GDS export, the <b>2019/09</b> subdirectory in the <b>gsfs://127.0.0.1:7789/2019/09/</b> directory specified by the <b>location</b> table is automatically created.</li><li>If <b>file_type</b> is set to <b>pipe</b>, GDS determines whether the target file to be imported or exported is a pipe file or a directory based on whether the last character in the URL is a slash (/). Example:<ul style="list-style-type: none"><li>In <b>gsfs://192.168.0.90:5000/a/b</b>, GDS identifies <b>b</b> as a pipe file.</li></ul></li></ul> |

| Parameter | Description                                                      | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           |                                                                  | <ul style="list-style-type: none"><li>– In <code>gsfs://192.168.0.90:5000/a/b/</code>, GDS identifies <b>b</b> as a directory and creates a pipe file in the directory.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| format    | Specifies the format of the data source file in a foreign table. | <p>Value range: <b>CSV</b>, <b>TEXT</b>, or <b>FIXED</b>. The default value is <b>TEXT</b>.</p> <ul style="list-style-type: none"><li>• In CSV files, escape sequences are processed as common strings. Therefore, linefeeds are processed as data.</li><li>• In TEXT files, escape sequences are processed as they are. Therefore, linefeeds are not processed as data.</li><li>• The FIXED file can process newline characters in data columns efficiently, but cannot process special characters very well.</li></ul> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• An escape sequence is a string starting with a backslash (\), including <code>\b</code> (backspace), <code>\f</code> (formfeed page break), <code>\n</code> (new line), <code>\r</code> (carriage return), <code>\t</code> (horizontal tab), <code>\v</code> (vertical tab), <code>\number</code> (octal number), and <code>\xnumber</code> (hexadecimal number). In TEXT files, strings are processed as they are. In files of other formats, strings are processed as data.</li><li>• <b>FIXED</b> is defined as follows: (<b>POSITION</b> must be specified for each column when <b>FIXED</b> is used.)<ol style="list-style-type: none"><li>1. The column length of each record is the same.</li><li>2. Spaces are used for column padding. Left padding is used for the numeric type and right padding is used for the char type.</li><li>3. No delimiters are used between columns.</li></ol></li></ul> |

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| header     | <p>Specifies whether a data file contains a table header.</p> <p>header is available only for CSV and FIXED files.</p> <p>When data is imported, if <b>header</b> is <b>on</b>, the first row of the data file will be identified as title row and ignored. If header is <b>off</b>, the first row is identified as data.</p> <p>When data is exported, if <b>header</b> is <b>on</b>, <b>fileheader</b> must be specified. <b>fileheader</b> is used to specify the export header file format. If header is <b>off</b>, the exported file does not include a title row.</p> | <b>true/on</b> or <b>false/off</b> . The default value is <b>false</b> or <b>off</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| fileheader | <p>Specifies a file that defines the content in the header for exported data. The file contains one row of data description of each column.</p> <p>For example, to add a header in a file containing product information, define the file as follows:</p> <p>The information of products.\n</p>                                                                                                                                                                                                                                                                              | <ul style="list-style-type: none"><li>• This parameter is available only when <b>header</b> is <b>on</b> or <b>true</b>. The file must be prepared in advance.</li><li>• In Remote mode, the definition file must be put to the working directory of GDS (the <b>-d</b> directory specified when starting the GDS).</li><li>• The definition file can contain only one row of title information, and end with a newline character. Excess rows will be discarded. (Title information cannot contain newline character).</li><li>• The length of the definition file including the newline character cannot exceed 1 MB.</li></ul> |

| Parameter           | Description                                                                                                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| out_filename_prefix | <p>Specifies the name prefix of the exported data file exported using GDS from a write-only foreign table.</p> <p>If <b>file_type</b> is set to <b>pipe</b>, the pipe file <b>dbName_schemaName_foreignTableName.pipe</b> is generated.</p> <p>If both <b>out_filename_prefix</b> and <b>location</b> specify a pipe name, the pipe name specified in <b>location</b> is used.</p> | <ul style="list-style-type: none"><li>• The prefix of the specified file name must be valid and compliant with the restrictions of the file system in the physical environment where the GDS is deployed. Otherwise, the file will fail to be created.<ul style="list-style-type: none"><li>– The file name prefix can contain only lowercase letters, uppercase letters, digits, and underscores (_).</li><li>– The prefix of the specified export file name cannot contain feature fields reserved for the Windows and Linux OS, including but not limited to:<br/>"con","aux","nul","prn","com0","com1","com2","com3","com4","com5","com6","com7","com8","com9","lpt0","lpt1","lpt2","lpt3","lpt4","lpt5","lpt6","lpt7","lpt8","lpt9"</li><li>– The total length of the absolute path consisting of the exported file prefix, the path specified by <b>GDS -d</b>, <b>.dat</b>, or <b>.pipe</b> should be as required by the file system where GDS is deployed.</li><li>– It is required that the prefix can be correctly parsed and identified by the receiver (including but not limited to the original database where it was exported) of the data file. Identify and modify the option that causes the file name resolution problem (if any).</li></ul></li><li>• To concurrently perform export jobs, do not use the same file name prefix for them. Otherwise, the exported files may overwrite each other or be lost in the OS or file system.</li></ul> |



| Parameter | Description                                                                                                                                                                                           | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| delimiter | Specifies the column delimiter of data, and uses the default delimiter if it is not set. The default delimiter of TEXT is a tab and that of CSV is a comma (,). No delimiter is used in FIXED format. | <p>The value of <b>delimiter</b> can be a multi-character delimiter whose length is less than or equal to 10 bytes.</p> <ul style="list-style-type: none"><li>• A delimiter cannot be <b>\r</b> or <b>\n</b>.</li><li>• A delimiter cannot be the same as the <b>null</b> value. The delimiter of CSV cannot be same as the <b>quote</b> value.</li><li>• The delimiter for the TEXT format data cannot contain any of the following characters: \.abcdefghijklmnopqrstuvwxyz0123456789.</li><li>• The data length of a single row should be less than 1 GB. If the delimiters are too long and there are too many rows, the length of valid data will be affected.</li><li>• You are advised to use a multi-character, such as the combination of the dollar sign (\$), caret (^), the ampersand (&amp;), or invisible characters, such as 0x07, 0x08, and 0x1b as the delimiter.</li><li>• For a multi-character delimiter, do not use the same characters, for example, ---.</li></ul> |
| quote     | Specifies which characters in a CSV source data file will be identified as quotation marks. The default value is a double quotation mark (").                                                         | <ul style="list-style-type: none"><li>• The quote parameter cannot be the same as the delimiter or null parameter.</li><li>• The <b>quote</b> parameter must be a single-byte character.</li><li>• Invisible characters are recommended as <b>quote</b> values, such as 0x07, 0x08, and 0x1b.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| escape    | Specifies which characters in a CSV source data file are escape characters. Escape characters can only be single-byte characters.<br><br>Default value: the same as the value of QUOTE                | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |

| Parameter                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| null                     | Specifies the string that represents a null value.                                                                                                                                                                                                                                                                                                                                                                                                                                 | <ul style="list-style-type: none"><li>• The default value is <b>\N</b> for the TEXT format.</li><li>• The default value for the CSV format is an empty string without quotation marks.</li><li>• The null value cannot be <b>\r</b> or <b>\n</b>. The maximum length is 100 characters.</li><li>• The null value cannot be the same as the delimiter or quote parameter.</li></ul> |
| blank_number_str_to_null | Converts a variable number of spaces of the numeric type ( <b>INT1OID</b> , <b>INT2OID</b> , <b>INT4OID</b> , <b>INT8OID</b> , <b>FLOAT4OID</b> , or <b>FLOAT8OID</b> ) in a data file to <b>NULL</b> . This parameter is supported only by clusters of version 8.1.3, 8.2.1.236, 9.1.0, or later.<br><br>For example, if this option is set to <b>true</b> , <b>NULL</b> is returned when <b>null</b> , <b>null ' ' (one space)</b> , or <b>null ' ' (two spaces)</b> is defined. | The value can be <b>true</b> or <b>false</b> . The default value is <b>false</b> , which means no conversion is done.                                                                                                                                                                                                                                                              |
| noescaping               | Specifies in TEXT format, whether to escape the backslash (\) and its following characters.                                                                                                                                                                                                                                                                                                                                                                                        | <b>noescaping</b> is available only for the TEXT format. <b>true/on</b> or <b>false/off</b> . The default value is <b>false</b> or <b>off</b> .                                                                                                                                                                                                                                    |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Value Range                                                                                                                                                                                                                                                                                                                                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| encoding  | <p>Specifies the encoding of a data file, that is, the encoding used to parse, check, and generate a data file. Its default value is the default <b>client_encoding</b> value of the current database.</p> <p>Before you import foreign tables, it is recommended that you set <b>client_encoding</b> to the file encoding format, or a format matching the character set of the file. Otherwise, unnecessary parsing and check errors may occur, leading to import errors, rollback, or even invalid data import. Before you import foreign tables, you are also advised to specify this parameter, because the export result using the default character set may not be what you expected.</p> <p>If this parameter is not specified when you create a foreign table, a warning message will be displayed on the client.</p> | <ul style="list-style-type: none"><li>Currently, GDS cannot parse or write in a file using multiple encoding formats during foreign table import or export.</li><li>Common encoding formats include UTF-8, GBK, and GB18030. GB18030 has two versions: GB18030 and GB18030_2022. GB18030_2022 is the latest national standard in China prepared to support Chinese characters.</li></ul> |

| Parameter    | Description                                                                                                                                                                                     | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| dataencoding | Specifies the actual encoding of data in tables in the latin1 database. Converts data format during GDS data import and export. This parameter is supported by version 8.2.0 or later clusters. | <b>GBK, GB18030, GB18030_2022, and UTF8.</b><br>Typical scenarios:<br><ol style="list-style-type: none"><li>To import the GB-encoded text data to the <b>latin1</b> database and convert the data to UTF-8. Set <b>dataencoding</b> to <b>UTF8</b> and <b>encoding</b> to <b>GB</b>.</li><li>To export GB-encoded data from the <b>latin1</b> database as a UTF8 data file, set <b>dataencoding</b> to <b>GB</b> and <b>encoding</b> to <b>UTF8</b>.</li></ol> <b>CAUTION</b> <ul style="list-style-type: none"><li>This parameter can be used only in <b>latin1</b> databases.</li><li>When this parameter is used, <b>encoding</b> must be <b>GBK, GB18030, GB18030_2022, or UTF8</b>.</li></ul>                                                                                                                                                                                                                                                                                                                                            |
| mode         | Specifies the data import policy during a specific data import process. DWS supports only the <b>Normal</b> mode.                                                                               | Valid value: <ul style="list-style-type: none"><li><b>Normal</b> (default): supports all file types (CSV, TEXT, FIXED). Enabling Gauss data service to help data import.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| eol          | Specifies the newline character style of the imported or exported data file.                                                                                                                    | Value range: multi-character newline characters within 10 bytes. Common newline characters include <b>\r</b> (0x0D), <b>\n</b> (0x0A), and <b>\r\n</b> (0x0D0A). Special newline characters include <b>\$</b> and <b>#</b> . <ul style="list-style-type: none"><li>The <b>eol</b> parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format for data import. For forward compatibility, the <b>eol</b> parameter can be set to <b>0x0D</b> or <b>0x0D0A</b> for data export in the CSV and FIXED formats.</li><li>If you want to export data in FIXED format as a single row, use the <b>eol</b> parameter and set it to an empty string ". However, it is important to note that this method of exporting data cannot be directly imported into the DWS database.</li><li>The value of the <b>eol</b> parameter cannot be the same as that of <b>DELIMITER</b> or <b>NULL</b>.</li><li>The value of the <b>eol</b> parameter cannot contain digits, letters, or periods (.).</li></ul> |

| Parameter          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| conflict_delimiter | <p>This parameter is generally used with the <b>compatible_illegal_chars</b> parameter. If a data file contains a truncated Chinese character, the truncated character and a delimiter will be encoded into another Chinese character due to inconsistent encoding between the foreign table and the database. As a result, the delimiter is masked and an error will be reported, indicating that there are missing fields.</p> <p>This parameter is used to avoid encoding a truncated character and a delimiter into another character.</p> | <p>Value range: <b>true</b>, <b>on</b>, <b>false</b>, and <b>off</b>. The default value is <b>false</b> or <b>off</b>.</p> <ul style="list-style-type: none"><li>• If the parameter is set to <b>true</b> or <b>on</b>, encoding a truncated character and a delimiter into another character is allowed.</li><li>• If the parameter is set to <b>false</b> or <b>off</b>, encoding a truncated character and a delimiter into another character is not allowed.</li></ul> <p>This parameter is disabled by default. It is recommended that you disable this parameter, because encoding a truncated character and a delimiter into another character is rarely required. If the parameter is enabled, the scenario may be incorrectly identified and thereby causing incorrect information imported to the table.</p>                                      |
| file_type          | <p>Specifies the type of the file to be imported or exported.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | <p>Value options: <b>normal</b>, <b>pipe</b>, and <b>interconn</b>. <b>normal</b> is the default value.</p> <ul style="list-style-type: none"><li>• If this parameter is set to <b>normal</b>, the file to be imported or exported is a common file.</li><li>• If this parameter is set to <b>pipe</b>, the file to be imported or exported is a named pipe.</li><li>• If the parameter is <b>interconn</b>, the current GDS foreign table is used for interconnected and compressed import and export. This parameter must be used together with the <b>gds_compress</b> parameter and is supported only by cluster 8.2.0 or later.</li></ul> <p><b>CAUTION</b></p> <p>If the value of <b>file_type</b> is set to <b>interconn</b>, GDS must be upgraded to 8.2.0 or later. Otherwise, the error message "ERROR: un-support format" will be displayed.</p> |

| Parameter        | Description                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| auto_create_pipe | <p>Specifies whether the GDS process automatically creates a named pipe.</p> <p>When configuring <b>auto_create_pipe</b>, you need to set <b>file_type</b> to <b>pipe</b> or <b>interconn</b>. Otherwise, the foreign table cannot be created.</p> | <p>Value options: <b>true</b>, <b>on</b>, <b>false</b>, and <b>off</b>. The default value is <b>true/on</b>.</p> <ul style="list-style-type: none"><li>• If this parameter is set to <b>true</b> or <b>on</b>, the GDS process is allowed to automatically create a named pipe.</li><li>• If this parameter is set to <b>false</b> or <b>off</b>, you need to manually create a named pipe.</li><li>• If <b>auto_create_pipe</b> is set to <b>false</b> and no pipe is specified during data import and export, the <i>database name_schema name_foreign table name.pipe</i> file will be opened. If a pipe has been specified, the specified pipe in the location will be opened. If the named pipe is not written by other programs or is not opened in write mode within the period specified by the <b>pipe-timeout</b> parameter, an error message is displayed indicating that the import or export task times out. If the file is not a pipe, an error is reported when the import or export task is executed.</li><li>• If <b>auto_create_pipe</b> is set to <b>true</b> and no pipe file is specified during data import and export, the <i>database name_schema name_foreign table name.pipe</i> file will be opened. If the file is a common file, an error is reported when the file is imported or exported. If the file is a pipe, the system automatically deletes the file and re-creates the named pipe.</li><li>• You can use the <b>location</b> parameter to specify the pipe during data export, for example, <b>location'gsfs://127.0.0.1:7789/aa.pipe</b>. When <b>auto_create_pipe</b> is set to <b>true</b>, GDS automatically creates the <b>aa.pipe</b> file in the data directory.</li></ul> |

| Parameter | Description                                                                                                                                                                                                                                                               | Value Range                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| del_pipe  | <p>Specifies whether to automatically delete the pipe file after the import or export task is complete.</p> <p>When configuring <b>del_pipe</b>, you need to set <b>file_type</b> to <b>pipe</b> or <b>interconn</b>. Otherwise, the foreign table cannot be created.</p> | <p>Value options: <b>true</b> or <b>on</b>; <b>false</b> or <b>off</b>. The default value is <b>true</b> or <b>on</b>.</p> <ul style="list-style-type: none"><li>• If this parameter is set to <b>true</b> or <b>on</b>, the GDS process will automatically delete a named pipe file.</li><li>• If this parameter is set to <b>false</b> or <b>off</b>, the GDS process will not delete a named pipe file.</li></ul> |

| Parameter                    | Description                                                                                                                                                                                                                                             | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>gds_compress</code>    | This parameter is used to reduce the network bandwidth for cluster interconnection when users use GDS pipe files. Currently, and only the Snappy algorithm is supported. This parameter is supported by version 8.2.0 or later clusters.                | <p>Value range: <b>snappy</b>. Currently, only the Snappy algorithm is supported, which means that data imported and exported through GDS pipes is compressed and decompressed using the Snappy algorithm.</p> <ul style="list-style-type: none"><li>When setting this parameter, <b>file_type</b> must be set to <b>interconn</b> and format to <b>CSV</b>. If this parameter is not set, by default, files are not compressed. Currently, this parameter can only be set to <b>snappy</b>. When using this function, ensure that the versions of the local cluster, remote cluster, and GDS are of the same version.</li><li>If the value of <b>file_type</b> is <b>interconn</b>, GDS must be upgraded to 8.2.0 or later. Otherwise, the error message "ERROR: unsupported format" is displayed.</li><li>This parameter reduces the network bandwidth between clusters and increases the cluster CPU load. You can determine whether to enable this parameter based on the site requirements.</li><li>When a GDS foreign table is used for interconnected data migration, to allow data to be compressed during migration, the GDS foreign table must support this parameter. You use interconnected foreign tables to compress and migrate data, or manually use a GDS foreign table to compress and migrate data at the network layer. Many parameters need to be set during manual procedure, you are advised to use interconnected foreign tables.</li><li>This parameter must be specified for foreign tables used for both export and import.</li></ul> |
| <code>preserve_blanks</code> | Specifies whether to retain the blank characters (including spaces, <code>\t</code> , <code>\v</code> , and <code>\f</code> ) at the end of each column during fixed-length import. This parameter is supported by version 8.2.0.100 or later clusters. | <p>Value range: <b>true</b>, <b>on</b>, <b>false</b>, and <b>off</b>. The default value is <b>false</b> or <b>off</b>.</p> <ul style="list-style-type: none"><li>If this parameter is set to <b>true</b> or <b>on</b>, the trailing <code>\t</code> is retained and pruning is not performed for column data of the BYTEAOID, CHAROID, NAMEOID, TEXTOID, BPCHAROID, VARCHAROID, NVARCHAR2OID or CSTRINGOID type.</li><li>If this parameter is set to <b>false</b> or <b>off</b>, blank characters (including spaces, <code>\t</code>, <code>\v</code>, and <code>\f</code>) at the end of all data types are pruned.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |



| Parameter                 | Description                                                                                                                                                                                                                                                                                                                                                                                    | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fix                       | Specifies the length of fixed format data. The unit is byte. This syntax is available only for READ ONLY foreign tables.                                                                                                                                                                                                                                                                       | Value range: Less than <b>1 GB</b> , and greater than or equal to the total length specified by <b>POSITION</b> (The total length is the sum of <b>offset</b> and <b>length</b> in the last column of the table definition.)                                                                                                                                                                                                                                                                                                                                                                                                      |
| out_fix_alignme<br>nt     | Specifies how the columns of types BYTEAOID, CHAROID, NAMEOID, TEXTOID, BPCHAROID, VARCHAROID, NVARCHAR2OID, and CSTRINGOID are aligned during fixed-length export.                                                                                                                                                                                                                            | Value range: <b>align_left</b> , <b>align_right</b><br>Default value: <b>align_right</b><br><b>CAUTION</b><br>The bytea data type must be in hexadecimal format (for example, \XXXX) or octal format (for example, \XXX\XXX\XXX). The data to be imported must be left-aligned (that is, the column data starts with either of the two formats instead of spaces). Therefore, if the exported file needs to be imported using a GDS foreign table and the file data length is less than that specified by the foreign table formatter, the exported file must be left aligned. Otherwise, an error is reported during the import. |
| out_fix_num_ali<br>gnment | Specifies how the columns of numeric types TINYINT, SMALLINT, INTEGER, BINARY_INTEGER, BIGINT, NUMERIC[(p[,s])], DECIMAL[(p[,s])], NUMBER[(p[,s])], REAL, FLOAT4, DOUBLE PRECISION, FLOAT8, FLOAT[(p)], BINARY_DOUBLE, DEC[(p[,s])], INTEGER[(p[,s])], SMALLSERIAL, SERIAL and BIGSERIAL are aligned during fixed-length export. This is supported only by clusters of version 9.1.0 or later. | Value range: <b>align_left</b> , <b>align_right</b><br>Default value: <b>align_left</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

| Parameter            | Description                                                                                         | Value Range                                                                                                                                                                                                                                                                           |
|----------------------|-----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| date_format          | Imports data of the DATE type. This syntax is available only for READ ONLY foreign tables.          | Value range: any valid DATE value. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .<br><b>CAUTION</b><br>If ORACLE is specified as the compatible database, the DATE format is TIMESTAMP. For details, see <a href="#">timestamp_format</a> below. |
| time_format          | Imports data of the TIME type. This syntax is available only for READ ONLY foreign tables.          | Value range: any valid TIME value. Time zones cannot be used. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                                                                                                                                     |
| timestamp_format     | Imports data of the TIMESTAMP type. This syntax is available only for READ ONLY foreign tables.     | Value range: any valid TIMESTAMP value. Time zones are not supported. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                                                                                                                             |
| smalldatetime_format | Imports data of the SMALLDATETIME type. This syntax is available only for READ ONLY foreign tables. | Value range: any valid SMALLDATETIME value. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                                                                                                                                                       |

**Table 21-185** OPTIONS fault tolerance parameters

| Parameter           | Description                                                                                                                 | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| fill_missing_fields | Specifies whether to generate an error message when the last column in a row in the source file is lost during data import. | Value range: <b>true</b> , <b>on</b> , <b>false</b> , and <b>off</b> . The default value is <b>false</b> or <b>off</b> . <ul style="list-style-type: none"><li>If this parameter is set to <b>true</b> or <b>on</b> and the last column of a data row in a data source file is lost, the column will be replaced with <b>NULL</b> and no error message will be generated.</li><li>If this parameter is set to <b>false</b> or <b>off</b> and the last column is missing, the following error information will be displayed:<br/>missing data for column "tt"</li></ul> |

| Parameter         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ignore_extra_data | Specifies whether to ignore excessive columns when the number of data source files exceeds the number of foreign table columns. This parameter is available during data import.                                                                                                                                                                                                                                                                                                                                                                         | <p>Value range: <b>true</b>, <b>on</b>, <b>false</b>, and <b>off</b>. The default value is <b>false</b> or <b>off</b>.</p> <ul style="list-style-type: none"><li>If this parameter is set to <b>true</b> or <b>on</b> and the number of data source files exceeds the number of foreign table columns, excessive columns will be ignored.</li><li>If this parameter is set to <b>false</b> or <b>off</b> and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:<br/>extra data after last expected column</li></ul> <p><b>CAUTION</b><br/>If the newline character at the end of the row is lost, setting the parameter to <b>true</b> will ignore data in the next row.</p> |
| reject_limit      | <p>Specifies the maximum number of data format errors allowed during a data import task. If the number of errors does not reach the maximum number, the data import task can still be executed.</p> <p><b>NOTE</b><br/>You are advised to replace this syntax with <b>PER NODE REJECT LIMIT 'value'</b>.</p> <p>Examples of data format errors include the following: a column is lost, an extra column exists, a data type is incorrect, and encoding is incorrect. Once a non-data format error occurs, the whole data import process is stopped.</p> | <p>Value range: a positive integer or <b>unlimited</b>. Enclose positive integer values with single quotation marks ('').</p> <p>If this parameter is not specified, an error message is returned immediately.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

| Parameter                             | Description                                                                                                                                                                                                                                 | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>compatible_illegal_chars</code> | <p>Specifies whether to tolerate invalid characters during data import and export. This syntax is valid for READ ONLY and WRITE ONLY foreign tables.</p> <p>Only clusters of version 8.1.3.331 or later support export fault tolerance.</p> | <p>Value range: <b>true</b>, <b>on</b>, <b>false</b>, and <b>off</b>. The default value is <b>false</b> or <b>off</b>.</p> <ul style="list-style-type: none"><li>• If the parameter is <b>true</b> or <b>on</b>, the database will accept invalid characters after converting them during import or export.</li><li>• If this parameter is set to <b>false</b> or <b>off</b>, the import process will be stopped if there are invalid characters and an error occurs.</li><li>• The rule of error tolerance when you import invalid characters is as follows:<ul style="list-style-type: none"><li>– <code>\0</code> is converted to a space.</li><li>– Other illegal characters are converted to question marks.</li><li>– If <code>compatible_illegal_chars</code> is set to <b>true</b> or <b>on</b>, the database will convert and accept the invalid characters. If <b>NULL</b>, <b>DELIMITER</b>, <b>QUOTE</b>, and <b>ESCAPE</b> are set to a spaces or question marks. Errors like "illegal chars conversion may confuse COPY escape 0x20" will be displayed to prompt user to modify parameter values that cause confusion, preventing import errors.</li><li>– Enabling error tolerance for foreign table export will result in invalid characters being exported as question marks (?), which can lead to inconsistencies between the exported and original data when imported back into the DWS database.</li></ul></li></ul> |
| <code>replace_illegal_chars</code>    | <p>Error tolerance parameter for importing invalid characters. This parameter is used to replace <code>\0</code> with a specified string. This syntax is available only for READ ONLY foreign tables.</p>                                   | <p>The restrictions on the string used to replace <code>\0</code> are as follows:</p> <ul style="list-style-type: none"><li>• <code>compatible_illegal_chars</code> must be set to <b>true</b> or <b>on</b>.</li><li>• <code>format</code> must be set to <b>TEXT</b> or <b>CSV</b>.</li><li>• A maximum of 10 characters is allowed.</li><li>• The string cannot be the same as the delimiter, or contain the delimiter, or contain the following characters: digits, letters, backslashes (<code>\</code>), and periods (<code>.</code>).</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

| Parameter                       | Description                                                                                                                                                                                                                                                                                                                                                                      | Value Range                                             |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| WITH<br>error_table_name        | Data format errors during import are recorded in the table specified by <b>error_table_name</b> . You can query this table after the import to obtain error details. This parameter is available only after <b>reject_limit</b> is set.<br><b>NOTE</b><br>To be compatible with PostgreSQL open source interfaces, you are advised to replace this syntax with <b>LOG INTO</b> . | A string, which must comply with the naming convention. |
| LOG<br>INTO<br>error_table_name | Data format errors during import are recorded in the table specified by <b>error_table_name</b> . You can query this table after the import to obtain error details.<br><b>NOTE</b><br>This parameter is available only after <b>PER NODE REJECT LIMIT</b> is set.                                                                                                               | A string, which must comply with the naming convention. |
| REMOTE LOG<br>'name'            | The data format error information is saved as files in GDS. <b>name</b> is the prefix of the error data file.                                                                                                                                                                                                                                                                    | -                                                       |

| Parameter                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range                                                                                                |
|-------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| PER<br>NODE<br>REJECT<br>LIMIT<br>'value' | <p>This parameter specifies the allowed number of data format errors on each DN during data import. If the number of errors exceeds the specified value on any DN, data import fails, an error is reported, and the system exits data import.</p> <p><b>CAUTION</b><br/>This syntax specifies the error tolerance of a single node.<br/>Examples of data format errors include the following: a column is lost, an extra column exists, a data type is incorrect, and encoding is incorrect. When a non-data format error occurs, the whole data scanning process is stopped.</p> | An <b>unlimited</b> integer. If this parameter is not specified, an error message is returned immediately. |

**Table 21-186** OPTIONS performance parameters

| Parameter     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| file_sequence | <p>Concurrently imports data in parallel through GDS foreign tables, to improve single-file import performance. This parameter is only used for data import.</p> <p>The parameter format is <b>file_sequence</b>'<i>total number of shards-current shard</i>'.<br/>Example:</p> <p><b>file_sequence '3-1'</b> indicates that the imported file is logically split into three shards and the data currently imported by the foreign table is the data on the first shard.</p> <p><b>file_sequence '3-2'</b> indicates that the imported file is logically split into three shards and the data currently imported by the foreign table is the data on the second shard.</p> <p><b>file_sequence '3-3'</b> indicates that the imported file is logically split into three shards and the data currently imported by the foreign table is the data on the third shard.</p> | <p>This parameter has the following constraints:</p> <ul style="list-style-type: none"><li>• A file can be split to a maximum of 8 shards.</li><li>• The number of currently imported shard should be less than or equal to the total number of split shards.</li><li>• Only CSV and TXT files can be imported.</li></ul> <p>When data is imported in parallel in CSV format, some shards fail to be imported in some scenarios because the CSV rules conflict with the GDS splitting logic. For details, see <a href="#">Exception Description When the file_sequence Parameter Is Used</a>.</p> |

## Examples

Create a foreign table **customer\_ft** to import data from GDS server 10.10.123.234 in TEXT format:

```
CREATE FOREIGN TABLE customer_ft
(
 c_customer_sk integer ,
 c_customer_id char(16) ,
 c_current_demo_sk integer ,
 c_current_demo_sk integer ,
 c_current_addr_sk integer ,
 c_first_ship_to_date_sk integer ,
 c_first_sales_date_sk integer ,
 c_salutation char(10) ,
 c_first_name char(20) ,
 c_last_name char(30) ,
 c_preferred_cust_flag char(1) ,
 c_birth_day integer ,
 c_birth_month integer ,
 c_birth_year integer ,
 c_birth_country varchar(20) ,
 c_login char(13) ,
 c_email_address char(50) ,
 c_last_review_date char(10)
)
 SERVER gsmpp_server
 OPTIONS
(
 location 'gsfs://10.10.123.234:5000/customer1*.dat',
 FORMAT 'TEXT' ,
 DELIMITER '|',
 encoding 'utf8',
 mode 'Normal')
READ ONLY;
```

Create a foreign table to import data from GDS servers 192.168.0.90 and 192.168.0.91 in TEXT format. Record errors that occur during data import in **foreign\_HR\_staffs\_ft**. A maximum of two data format errors are allowed during the data import.

```
CREATE FOREIGN TABLE foreign_HR_staffs_ft
(
 staff_ID NUMBER(6) ,
 FIRST_NAME VARCHAR2(20),
 LAST_NAME VARCHAR2(25),
 EMAIL VARCHAR2(25),
 PHONE_NUMBER VARCHAR2(20),
 HIRE_DATE DATE,
 employment_ID VARCHAR2(10),
 SALARY NUMBER(8,2),
 COMMISSION_PCT NUMBER(2,2),
 MANAGER_ID NUMBER(6),
 section_ID NUMBER(4)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*', format
'TEXT', delimiter 'E'\x08', null '', reject_limit '2') WITH err_HR_staffs_ft;
```

Create a foreign table to import all files in the **input\_data** directory in CSV format.

```
CREATE FOREIGN TABLE foreign_HR_staffs_ft1
(
 staff_ID NUMBER(6) ,
 FIRST_NAME VARCHAR2(20),
 LAST_NAME VARCHAR2(25),
 EMAIL VARCHAR2(25),
 PHONE_NUMBER VARCHAR2(20),
 HIRE_DATE DATE,
 employment_ID VARCHAR2(10),
```



```
SALARY NUMBER(8,2),
COMMISSION_PCT NUMBER(2,2),
MANAGER_ID NUMBER(6),
section_ID NUMBER(4)
) SERVER gsmpp_server OPTIONS (location 'file:///input_data/', format 'csv', quote E'\x08', mode 'private',
delimiter ',') WITH err_HR_staffs_ft1;
```

Create a foreign table to export data to the **output\_data** directory in CSV format.

```
CREATE FOREIGN TABLE foreign_HR_staffs_ft2
(
 staff_ID NUMBER(6) ,
 FIRST_NAME VARCHAR2(20),
 LAST_NAME VARCHAR2(25),
 EMAIL VARCHAR2(25),
 PHONE_NUMBER VARCHAR2(20),
 HIRE_DATE DATE,
 employment_ID VARCHAR2(10),
 SALARY NUMBER(8,2),
 COMMISSION_PCT NUMBER(2,2),
 MANAGER_ID NUMBER(6),
 section_ID NUMBER(4)
) SERVER gsmpp_server OPTIONS (location 'file:///output_data/', format 'csv', quote E'\x08', delimiter '|',
header 'on') WRITE ONLY;
```

## Exception Description When the file\_sequence Parameter Is Used

When data is imported in parallel in CSV format, some shards fail to be imported in the following scenario because the CSV rules conflict with the GDS splitting logic:

**Scenario:** A CSV file contains a newline character that is not escaped, the newline character is contained in the character specified by **quote**, and the data of this line is in the first row of the logical shard.

**Example:** If you import the **big.csv** file in parallel, the following information is displayed:

```
--id, username, address
10001,"customer1 name","Rose District"
10002,"customer2 name","
23 Road Rose
District NewCity"
10003,"customer3 name","NewCity"
```

After the file is split into two shards, the content of the first shard is as follows:

```
10001,"customer1 name","Rose District"
10002,"customer2 name","
23
```

The content of the second shard is as follows:

```
Road Rose
District NewCity"
10003,"customer3 name","NewCity"
```

The newline character after the first line of the second shard is contained between double quotation marks. As a result, GDS cannot determine whether the newline character is a newline character in the field or a separator in the line. Therefore, two data records on the first shard are successfully imported, but the second shard fails to be imported.

## Helpful Links

[ALTER FOREIGN TABLE \(GDS Import and Export\)](#), [DROP FOREIGN TABLE](#)

### 21.13.4.11 ALTER FOREIGN TABLE (for HDFS or OBS)

#### Function

Modifies an HDFS or OBS foreign table.

A foreign table is a virtual table within a database or big data platform that does not physically store data. Instead, it serves as a mapping to data residing in external storage systems, such as HDFS or OBS. This mapping is established using metadata, which includes the table structure, the location of the data, and its file format. It enables users to directly access or operate external data using standard SQL statements. A foreign table serves as a data proxy that logically maps data in external file systems to table structures that can be identified by databases.

Foreign tables are designed to break data silos. Foreign tables allow database users to access external system data in a familiar way through metadata mapping and export data to external systems. In this way, data can be efficiently collaborated, analyzed, and shared across systems, especially in big data scenarios, significantly reducing data movement costs and improving collaboration efficiency.

#### Precautions

None

#### Syntax

- Set the attributes of a foreign table.  

```
ALTER FOREIGN TABLE [IF EXISTS] table_name
 OPTIONS ({[ADD | SET | DROP] option ['value']} [, ...]);
```
- Set the owner of the foreign table.  

```
ALTER FOREIGN TABLE [IF EXISTS] tablename
 OWNER TO new_owner;
```
- Update a foreign table column.  

```
ALTER FOREIGN TABLE [IF EXISTS] table_name
 MODIFY ({ column_name data_type | column_name [CONSTRAINT constraint_name] NOT NULL
 [ENABLE] | column_name [CONSTRAINT constraint_name] NULL } [, ...]);
```
- Modify the columns of a foreign table.  

```
ALTER FOREIGN TABLE [IF EXISTS] tablename
 action [, ...];
```

The **action** syntax is as follows:

```
ALTER [COLUMN] column_name [SET DATA] TYPE data_type
| ALTER [COLUMN] column_name { SET | DROP } NOT NULL
| ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer
| ALTER [COLUMN] column_name OPTIONS ({[ADD | SET | DROP] option ['value']} [, ...])
| MODIFY column_name data_type
| MODIFY column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE]
| MODIFY column_name [CONSTRAINT constraint_name] NULL
```

For details, see [ALTER TABLE](#).

- Add an informational constraint for a foreign table.  

```
ALTER FOREIGN TABLE [IF EXISTS] tablename
 ADD [CONSTRAINT constraint_name]
```

```
{ PRIMARY KEY | UNIQUE } (column_name)
[NOT ENFORCED [ENABLE QUERY OPTIMIZATION | DISABLE QUERY OPTIMIZATION] |
ENFORCED];
```

For details about the parameters for adding an informational constraint, see [Parameter Description](#).

- Remove a foreign table informational constraint.

```
ALTER FOREIGN TABLE [IF EXISTS] tablename
DROP CONSTRAINT constraint_name ;
```

- Set the row-level access control switch for foreign tables (supported by cluster versions 9.1.0 and later).

```
ALTER FOREIGN TABLE [IF EXISTS] tablename
ENABLE | DISABLE | FORCE | NO FORCE ROW LEVEL SECURITY
```

## Parameter Description

**Table 21-187** ALTER FOREIGN TABLE (For HDFS or OBS) parameters

| Parameter       | Description                                                                        | Value Range                               |
|-----------------|------------------------------------------------------------------------------------|-------------------------------------------|
| IF EXISTS       | Sends a message instead of an error if the specified foreign table does not exist. | -                                         |
| tablename       | Specifies the name of an existing foreign table to be modified.                    | Name of an existing foreign table.        |
| new_owner       | Specifies the new owner of the foreign table.                                      | A string that indicates a valid username. |
| data_type       | Specifies the new data type of an existing column.                                 | A string.                                 |
| constraint_name | Specifies the name of a constraint to be added or deleted.                         | A string.                                 |
| column_name     | Specifies an existing column name.                                                 | A string.                                 |

For how to modify other parameters in a foreign table, see [Parameter Description](#) in **ALTER TABLE**.

## Examples

- Create **HDFS\_Server** and set the corresponding foreign data wrapper to **HDFS\_FDW** or **DFS\_FDW**.

```
CREATE SERVER hdfs_server FOREIGN DATA WRAPPER HDFS_FDW OPTIONS (address
'10.10.0.100:25000,10.10.0.101:25000',hdfscfgpath '/opt/hadoop_client/HDFS/hadoop/etc/
hadoop',type'HDFS');
```

- Create an HDFS foreign table that does not contain partition columns.

```
DROP FOREIGN TABLE IF EXISTS ft_region;
CREATE FOREIGN TABLE ft_region
```

```
(
 R_REGIONKEY INT4,
 R_NAME TEXT,
 R_COMMENT TEXT
)
SERVER
 hdfs_server
OPTIONS
(
 FORMAT 'orc',
 encoding 'utf8',
 FOLDERNAME '/user/hive/warehouse/gauss.db/region_orc11_64stripe/'
)
DISTRIBUTE BY
 roundrobin;
```

3. Change the type of the **r\_name** column in the foreign table **ft\_region** to **text**.  
ALTER FOREIGN TABLE ft\_region ALTER r\_name TYPE TEXT;
4. Mark the **r\_name** column of the foreign table **ft\_region** as **not null**.  
ALTER FOREIGN TABLE ft\_region ALTER r\_name SET NOT NULL;

Helpful Links

[CREATE FOREIGN TABLE \(SQL on OBS or Hadoop\)](#) and [DROP FOREIGN TABLE](#)

21.13.4.12 ALTER FOREIGN TABLE (GDS Import and Export)

Function

Modifies a GDS foreign table.

Precautions

None

Syntax

- Set the attributes of a foreign table.  
ALTER FOREIGN TABLE [ IF EXISTS ] table\_name  
 OPTIONS ( {[ ADD | SET | DROP ] option ['value']}[, ... ]);
- Set a new owner.  
ALTER FOREIGN TABLE [ IF EXISTS ] tablename  
 OWNER TO new\_owner;
- Set the row-level access control switch for foreign tables (supported by cluster versions 9.1.0 and later).  
ALTER FOREIGN TABLE [ IF EXISTS ] tablename  
 ENABLE | DISABLE | FORCE | NO FORCE ROW LEVEL SECURITY

Parameter Description

Table 21-188 ALTER FOREIGN TABLE (for GDS import and export) parameters

| Parameter | Description                                                                | Value Range |
|-----------|----------------------------------------------------------------------------|-------------|
| IF EXISTS | Sends a message instead of an error if the specified index does not exist. | -           |

| Parameter  | Description                                                     | Value Range                                                                           |
|------------|-----------------------------------------------------------------|---------------------------------------------------------------------------------------|
| table_name | Specifies the name of an existing foreign table to be modified. | Name of an existing foreign table.                                                    |
| option     | Specifies the name of the option to be modified.                | See <a href="#">Parameter Description</a> in <b>CREATE FOREIGN TABLE</b> for details. |
| value      | Specifies the new value of <b>option</b> .                      | -                                                                                     |
| new_owner  | Specifies the new owner of the foreign table.                   | A string that indicates a valid username.                                             |

## Examples

Create a foreign table **customer\_ft** to import data from GDS server 10.10.123.234 in TEXT format.

```
DROP FOREIGN TABLE IF EXISTS customer_ft;
CREATE FOREIGN TABLE customer_ft
(
 c_customer_sk integer ,
 c_customer_id char(16) ,
 c_current_demo_sk integer ,
 c_current_demo_sk integer ,
 c_current_addr_sk integer ,
 c_first_ship_date_sk integer ,
 c_first_sales_date_sk integer ,
 c_salutation char(10) ,
 c_first_name char(20) ,
 c_last_name char(30) ,
 c_preferred_cust_flag char(1) ,
 c_birth_day integer ,
 c_birth_month integer ,
 c_birth_year integer ,
 c_birth_country varchar(20) ,
 c_login char(13) ,
 c_email_address char(50) ,
 c_last_review_date char(10)
)
SERVER gsmpp_server
OPTIONS
(
 location 'gsfs://10.10.123.234:5000/customer1*.dat',
 FORMAT 'TEXT' ,
 DELIMITER '|',
 encoding 'utf8',
 mode 'Normal')READ ONLY;
```

Modify the **customer\_ft** attribute of the foreign table. Delete the **mode** option.

```
ALTER FOREIGN TABLE customer_ft options(drop mode);
```

## Helpful Links

[CREATE FOREIGN TABLE \(for GDS Import and Export\)](#) and [DROP FOREIGN TABLE](#)

### 21.13.4.13 DROP FOREIGN TABLE

#### Function

**DROP FOREIGN TABLE** deletes a specified foreign table.

#### Precautions

**DROP FOREIGN TABLE** forcibly deletes a specified table. After a table is deleted, any indexes that exist for the table will be deleted. The functions and stored procedures used in this table cannot be run.

#### Syntax

```
DROP FOREIGN TABLE [IF EXISTS]
table_name [, ...] [CASCADE | RESTRICT];
```

#### Parameter Description

**Table 21-189** DROP FOREIGN TABLE parameters

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                                                                                    |
|--------------------|-----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF EXISTS          | Sends a message instead of an error if the specified foreign table does not exist.                  | -                                                                                                                                                                                                                                                                                              |
| table_name         | Specifies the name of the foreign table to be deleted.                                              | An existing table name.                                                                                                                                                                                                                                                                        |
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: automatically deletes all objects (such as views) that depend on the foreign table to be deleted.</li><li>• <b>RESTRICT</b>: refuses to delete the foreign table if any objects depend on it. This is the default behaviour.</li></ul> |

#### Examples

Delete the foreign table named **customer\_ft**:

```
DROP FOREIGN TABLE customer_ft;
```

#### Helpful Links

[ALTER FOREIGN TABLE \(GDS Import and Export\)](#), [ALTER FOREIGN TABLE \(for HDFS or OBS\)](#), [CREATE FOREIGN TABLE \(for GDS Import and Export\)](#), [CREATE FOREIGN TABLE \(SQL on OBS or Hadoop\)](#)

## 21.13.4.14 CREATE SERVER

### Function

**CREATE SERVER** creates an external server.

An external server stores the HDFS cluster information, OBS server information, DLI connection information, or other homogeneous cluster information.

### Precautions

By default, only the system administrator can create a foreign server. Otherwise, creating a server requires USAGE permission on the foreign-data wrapper being used. The syntax is as follows:

```
GRANT USAGE ON FOREIGN DATA WRAPPER fdw_name TO username;
```

**fdw\_name** is the name of **FOREIGN DATA WRAPPER**, and **username** is the user name of creating **SERVER**.

### Syntax

```
CREATE SERVER server_name
 FOREIGN DATA WRAPPER fdw_name
 OPTIONS ({ option_name 'value' } [, ...]) ;
```

### Parameter Description

**Table 21-190** CREATE SERVER parameters

| Parameter                        | Description                                                                             | Value Range                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| server_name                      | Name of the foreign server to be created. The server name must be unique in a database. | A string, which must contain up to 63 characters and comply with the <a href="#">identifier naming conventions</a> .                                                                                                                                                                                                                                                                                                   |
| FOREIGN DATA WRAPPER<br>fdw_name | Specifies the name of the foreign data wrapper.                                         | <b>fdw_name</b> is the data wrapper created by the system during database initialization. <ul style="list-style-type: none"><li>For an HDFS cluster, <b>fdw_name</b> can be <b>hdfs_fdw</b> or <b>dfs_fdw</b>.</li><li>For other homogeneous clusters, <b>fdw_name</b> is <b>gc_fdw</b>. For data import and export, the GDS foreign table uses <b>gsmpp_server</b>, and <b>fdw_name</b> is <b>dist_fdw</b>.</li></ul> |
| OPTIONS                          | Specifies the parameters of foreign servers.                                            | For details about the supported parameters, see <a href="#">Table 21-191</a> .                                                                                                                                                                                                                                                                                                                                         |

**Table 21-191** OPTIONS parameters

| Parameter | Description                                                           | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|-----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| address   | Specifies the IP address of the OBS service endpoint or HDFS cluster. | <p>OBS: Specifies the endpoint of the OBS service.</p> <p>HDFS: Specifies the IP address and port number of a NameNode (metadata node) in the HDFS cluster, or the IP address and port number of a CN in other homogeneous clusters.</p> <p>HDFS NameNodes are deployed in primary/secondary mode for HA. Add the addresses of the primary and secondary NameNodes to <b>ADDRESS</b>. When accessing HDFS, DWS dynamically checks for the primary NameNode.</p> <p>If the HDFS is in federation mode, you can add all router addresses to the <b>address</b> value. When DWS accesses the HDFS service, the system dynamically and randomly searches for the active router.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• The <b>address</b> option must exist. In the cross-cluster interconnection scenario, only one <b>address</b> option can be set.</li><li>• If the server type is DLI, the address is the OBS address stored on DLI.</li><li>• If the HDFS is in federation mode, that is, <b>fed 'rbf'</b>, address can be set to multiple groups of IP addresses and ports, corresponding to the address of the HDFS router.</li></ul> |



| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Value Range                                                                                                                                                                                                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hdfscfgpath | <p>This parameter is available only when <b>type</b> is <b>HDFS</b>.</p> <p>You can set the <b>hdfscfgpath</b> parameter to specify the HDFS configuration file path. DWS accesses the HDFS cluster based on the connection configuration mode and security mode specified in the HDFS configuration file stored in that path. If the HDFS cluster is connected in non-secure mode, data transmission encryption is not supported.</p> <p>If the <b>address option</b> is not specified, the address specified by <b>hdfscfgpath</b> in the configuration file is used by default.</p> | -                                                                                                                                                                                                                                                                                                                |
| fed         | Indicates that <b>dfs_fdw</b> is connected to HDFS in federation mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <p>The value <b>rbf</b> indicates that HDFS uses the RBF mode.</p> <p><b>NOTE</b><br/>This feature is supported in 8.1.2 or later.</p>                                                                                                                                                                           |
| encrypt     | Specifies whether data is encrypted. This parameter is available only when <b>type</b> is <b>OBS</b> . The default value is <b>off</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                               | <ul style="list-style-type: none"><li>• <b>on</b> indicates that data is encrypted.</li><li>• <b>off</b> indicates that data is not encrypted.</li></ul>                                                                                                                                                         |
| access_key  | Specifies the access key (AK) (obtained by users from the OBS console) used for the OBS access protocol. When you create a foreign table, its AK value is encrypted and saved to the metadata table of the database.                                                                                                                                                                                                                                                                                                                                                                   | <ul style="list-style-type: none"><li>• If <b>FOREIGN DATA WRAPPER</b> is set to <b>dfs_fdw</b>, this parameter can be set only when type is set to OBS.</li><li>• For clusters of version 8.2.0 or later, this parameter can be specified when <b>FOREIGN DATA WRAPPER</b> is set to <b>dist_fdw</b>.</li></ul> |

| Parameter         | Description                                                                                                                                                                                                                                                                                                                           | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| secret_access_key | Indicates the secret access key (SK) (obtained by users from the OBS page) used for the OBS access protocol. When you create a foreign table, its SK value is encrypted and saved to the metadata table of the database.                                                                                                              | <ul style="list-style-type: none"><li>• If <b>FOREIGN DATA WRAPPER</b> is set to <b>dfs_fdw</b>, this parameter can be set only when type is set to OBS.</li><li>• For clusters of 8.2.0 or later, this parameter can be specified when <b>FOREIGN DATA WRAPPER</b> is set to <b>dist_fdw</b>.</li></ul>                                                                                                                                            |
| security_token    | Corresponds to the <b>SecurityToken</b> value of the temporary security credential in IAM. A temporary AK, a temporary SK, and a temporary security token form a temporary security credential. The temporary security credential is valid for no more than 24 hours. This parameter is supported by version 8.2.0 or later clusters. | <ul style="list-style-type: none"><li>• If <b>FOREIGN DATA WRAPPER</b> is set to <b>dfs_fdw</b>, this parameter can be set only when type is set to OBS.</li><li>• For clusters of 8.2.0 or later, this parameter can be specified when <b>FOREIGN DATA WRAPPER</b> is set to <b>dist_fdw</b>.</li><li>• When this parameter is used, <b>access_key</b> and <b>secret_access_key</b> correspond to the temporary AK and SK, respectively.</li></ul> |
| type              | Specifies the <b>dfs_fdw</b> connection type.                                                                                                                                                                                                                                                                                         | <ul style="list-style-type: none"><li>• <b>OBS</b> indicates that OBS is connected.</li><li>• <b>HDFS</b> indicates that HDFS is connected.</li><li>• <b>DLI</b> indicates that DLI is connected.</li></ul>                                                                                                                                                                                                                                         |
| dli_address       | Specifies the endpoint of the DLI service. This parameter is available only when <b>type</b> is <b>DLI</b> .                                                                                                                                                                                                                          | -                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

| Parameter             | Description                                                                                                                                                                                                                                                                                   | Value Range |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| dli_access_key        | Specifies the access key (AK) (obtained by users from the DLI console) used for the DLI access protocol. When you create a foreign table, its AK value is encrypted and saved to the metadata table of the database. This parameter is available only when <b>type</b> is <b>DLI</b> .        | -           |
| dli_secret_access_key | Specifies the secret access key (SK) (obtained by users from the DLI console) used for the DLI access protocol. When you create a foreign table, its SK value is encrypted and saved to the metadata table of the database. This parameter is available only when <b>type</b> is <b>DLI</b> . | -           |
| dbname                | Specifies the database name of a remote cluster to be connected. This parameter is used for collaborative analysis and cross-cluster interconnection.                                                                                                                                         | -           |
| username              | Specifies the username of a remote cluster to be connected. This parameter is used for collaborative analysis and cross-cluster interconnection.                                                                                                                                              | -           |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| password  | <p>Specifies the password of a remote cluster to be connected. This parameter is used for collaborative analysis and cross-cluster interconnection.</p> <p><b>NOTE</b></p> <p>When an on-premises cluster is migrated to the cloud, the password in the server configuration exported from the on-premises cluster is in ciphertext. The encryption and decryption keys of the on-premises cluster are different from those of the cloud cluster. Therefore, if <b>CREATE SERVER</b> is executed on the cloud cluster, the execution fails and a decryption failure error is reported. In this case, you need to manually change the password in <b>CREATE SERVER</b> to a plaintext password.</p> | -           |
| syncsrv   | <p>This parameter is used only for cross-cluster interconnection and indicates the GDS service used during data synchronization. The method for setting this parameter is the same as that for setting the <b>location</b> attribute of the GDS foreign table.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                 | -           |

## Examples

Create the **hdfs\_server** server, in which **hdfs\_fdw** is the built-in foreign data wrapper. (The values of **address** and **hdfscfgpath** are for reference only. Change them as required.)

```
CREATE SERVER hdfs_server FOREIGN DATA WRAPPER HDFS_FDW OPTIONS
(address '10.10.0.100:25000,10.10.0.101:25000',
 hdfscfgpath '/opt/hadoop_client/HDFS/hadoop/etc/hadoop',
 type 'HDFS'
);
```

Create the **obs\_server** server, in which **dfs\_fdw** is the foreign-data wrapper.

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER DFS_FDW OPTIONS (
 address 'obs.example.com',
 access_key 'xxxxxxxx',
 secret_access_key 'yyyyyyyyyyyyyy',
 type 'obs'
);
```

Create the **dli\_server** server, in which **dfs\_fdw** is the foreign-data wrapper.

```
CREATE SERVER dli_server FOREIGN DATA WRAPPER DFS_FDW OPTIONS (
 address 'obs.example.com',
 access_key 'xxxxxxxx',
 secret_access_key 'yyyyyyyyyyyyyy',
 type 'dli',
 dli_address 'dli.example.com',
 dli_access_key 'xxxxxxxx',
 dli_secret_access_key 'yyyyyyyyyyyyyy'
);
```

Create another server in the homogeneous cluster, where **gc\_fdw** is the foreign data wrapper in the database.

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS
(address '10.10.0.100:25000,10.10.0.101:25000',
 dbname 'test',
 username 'test',
 password 'xxxxxxx'
);
```

Create a server whose **FOREIGN DATA WRAPPER** is **dist\_fdw** for importing and exporting text data on OBS.

```
CREATE SERVER import_server FOREIGN DATA WRAPPER DIST_FDW OPTIONS
(
 access_key 'ak_string',
 secret_access_key 'sk_string'
);
```

## Helpful Links

[ALTER SERVER](#), [DROP SERVER](#)

### 21.13.4.15 ALTER SERVER

#### Function

Adds, modifies, and deletes the definition of a foreign server.

Existing foreign servers can be queried from the **pg\_foreign\_server** system catalog.

#### Precautions

- Only the owner of the server has the permission to run the **ALTER SERVER** command. System administrators have this permission by default.
- To change the server owner, you must own the foreign server, belong directly or indirectly to the new owner's role, and have the **USAGE** permission on the server's foreign-data wrapper.

## Syntax

- Change the parameters for an external server.  

```
ALTER SERVER server_name [VERSION 'new_version']
[OPTIONS ({[ADD | SET | DROP] option ['value']} [, ...])];
```
- Change the owner of an external server.  

```
ALTER SERVER server_name
OWNER TO new_owner;
```
- Change the name of an external service.  

```
ALTER SERVER server_name
RENAME TO new_name;
```
- Refresh the HDFS configuration file.  

```
ALTER SERVER server_name REFRESH OPTIONS;
```

## Parameter Description

Table 21-192 ALTER SERVER parameters

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                               | Value Range                                                                                                            |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| server_name | Specifies the name of the external service to be modified.                                                                                                                                                                                                                                                                                                                                                | Name of an existing external server.                                                                                   |
| new_version | Specifies the new version of the external server.                                                                                                                                                                                                                                                                                                                                                         | -                                                                                                                      |
| OPTIONS     | Specifies options for the server.<br><b>ADD</b> , <b>SET</b> , and <b>DROP</b> specify the operations to be performed. If the <b>OPTIONS</b> option is not specified, the <b>ADD</b> operation is performed by default.<br><br>Currently, only <b>SET</b> is supported on an HDFS server. <b>ADD</b> and <b>DROP</b> are not supported. The syntax for SET and DROP operations is retained for later use. | <b>option</b> and <b>value</b> are corresponding operation parameters. For details, see <a href="#">Table 21-193</a> . |
| new_owner   | Specifies the new owner of the external server.                                                                                                                                                                                                                                                                                                                                                           | Valid username.                                                                                                        |
| new_name    | Specifies the new name of the external server.                                                                                                                                                                                                                                                                                                                                                            | A string compliant with the <a href="#">identifier naming rules</a> .                                                  |

| Parameter          | Description                                                                                                                                                                                                                                                                                                                               | Value Range |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| REFRESH<br>OPTIONS | <p>Refreshes the HDFS configuration file cache information. This command is executed when the configuration file changes. If this command is not executed, an access error may occur.</p> <p>This parameter is used to update the HDFS server configuration cache and cannot be concurrently accessed by related HDFS foreign tables.</p> | -           |

**Table 21-193** Parameters for modifying **OPTIONS** supported by external servers

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Value Range |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| address     | <p>Specifies the endpoint of the OBS service.</p> <p>Specifies the IP address and port number of the primary and standby nodes of the HDFS cluster.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• <b>address</b> is mandatory for HDFS servers. Therefore, <b>ADD</b> and <b>DROP</b> operations are not supported.</li><li>• <b>address</b> only supports IPv4 addresses in dot-decimal notation, and an address string cannot contain spaces. Groups of addresses are separated by commas (,). An IP address and a port number are separated by a colon (:). You are advised to configure two IP address and port pairs in an HDFS cluster. One is used as the socket address of the primary HDFS NameNode and another is used as that of the secondary HDFS NameNode.</li><li>• If the server type is DLI, the address is the OBS address stored on DLI.</li></ul> | -           |
| hdfscfgpath | <p>Specifies the HDFS cluster configuration file.</p> <p><b>NOTE</b></p> <ul style="list-style-type: none"><li>• If HDFS is in security mode, <b>hdfscfgpath</b> is mandatory.</li><li>• If you set <b>hdfscfgpath</b>, you can only set one value for <b>path</b>.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | -           |

| Parameter             | Description                                                                                                                                                                                                                                                                                           | Value Range                                                                                                                                              |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| fed                   | Indicates that <b>dfs_fdw</b> is connected to HDFS in federation mode.<br><br>The value <b>rbf</b> indicates that HDFS uses the Router-based Federation (RBF) mode.<br><b>NOTE</b><br>This parameter is supported in 8.1.2 or later. In 8.0.0, this parameter is supported only in 8.0.0.10 or later. | -                                                                                                                                                        |
| encrypt               | Specifies whether data is encrypted. This parameter is available only when <b>type</b> is <b>OBS</b> . The default value is <b>off</b> .                                                                                                                                                              | <ul style="list-style-type: none"><li>• <b>on</b> indicates that data is encrypted.</li><li>• <b>off</b> indicates that data is not encrypted.</li></ul> |
| access_key            | Indicates the access key (AK) (obtained by users from the OBS page) used for the OBS access protocol. When you create a foreign table, its AK value is encrypted and saved to the metadata table of the database. This parameter is available only when <b>type</b> is <b>OBS</b> .                   | -                                                                                                                                                        |
| secret_access_key     | Indicates the secret access key (SK) (obtained by users from the OBS page) used for the OBS access protocol. When you create a foreign table, its SK value is encrypted and saved to the metadata table of the database. This parameter is available only when <b>type</b> is <b>OBS</b> .            | -                                                                                                                                                        |
| dli_address           | Specifies the endpoint of the DLI service. This parameter is available only when <b>type</b> is <b>DLI</b> .                                                                                                                                                                                          | -                                                                                                                                                        |
| dli_access_key        | Specifies the access key (AK) (obtained by users from the DLI console) used for the DLI access protocol. When you create a foreign table, its AK value is encrypted and saved to the metadata table of the database. This parameter is available only when <b>type</b> is <b>DLI</b> .                | -                                                                                                                                                        |
| dli_secret_access_key | Specifies the secret access key (SK) (obtained by users from the DLI console) used for the DLI access protocol. When you create a foreign table, its SK value is encrypted and saved to the metadata table of the database. This parameter is available only when <b>type</b> is <b>DLI</b> .         | -                                                                                                                                                        |



| Parameter | Description                                                                                                                                                                                                                                                                                                   | Value Range |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| region    | Indicates the IP address or domain name of the OBS server. This parameter is available only when <b>type</b> is <b>OBS</b> .                                                                                                                                                                                  | -           |
| dbname    | Specifies the database name of a remote cluster to be connected. This parameter is used for collaborative analysis and cross-cluster interconnection.                                                                                                                                                         | -           |
| username  | Specifies the username of a remote cluster to be connected. This parameter is used for collaborative analysis and cross-cluster interconnection.                                                                                                                                                              | -           |
| password  | Specifies the password of a remote cluster to be connected. This parameter is used for collaborative analysis and cross-cluster interconnection.                                                                                                                                                              | -           |
| syncsrv   | This parameter is used only for cross-cluster interconnection and indicates the GDS service used during data synchronization. The method for setting this parameter is the same as that for setting the <b>location</b> attribute of the GDS foreign table. This feature is supported only in 8.1.2 or later. | -           |

## Examples

Create the **hdfs\_server** external server, in which **hdfs\_fdw** is the built-in foreign data wrapper.

```
CREATE SERVER hdfs_server FOREIGN DATA WRAPPER HDFS_FDW OPTIONS
(address '10.10.0.100:25000,10.10.0.101:25000',
 hdfsfgpath '/opt/hadoop_client/HDFS/hadoop/etc/hadoop',
 type 'HDFS'
);
```

Modify the **address** option of the external server **hdfs\_server**.

```
ALTER SERVER hdfs_server OPTIONS (SET address '10.10.0.110:25000,10.10.0.120:25000');
```

Modify the **hdfsfgpath** option of the external server **hdfs\_server**.

```
ALTER SERVER hdfs_server OPTIONS (SET hdfsfgpath '/opt/bigdata/hadoop');
```

## Helpful Links

[CREATE SERVER](#), [DROP SERVER](#)

## 21.13.4.16 DROP SERVER

### Function

**DROP SERVER** deletes an existing data server.

### Precautions

Only the server owner can delete a server.

### Syntax

```
DROP SERVER [IF EXISTS] server_name [{CASCADE | RESTRICT}] ;
```

### Parameter Description

**Table 21-194** DROP SERVER parameters

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                           |
|--------------------|-----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF EXISTS          | Sends a notice instead of an error if the specified server does not exist.                          | -                                                                                                                                                                                                                                     |
| server_name        | Specifies the name of the server to be deleted.                                                     | An existing server name.                                                                                                                                                                                                              |
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: automatically drops objects that depend on the server to be deleted.</li><li>• <b>RESTRICT</b> (default): refuses to delete the server if any objects depend on it.</li></ul> |

### Examples

Delete **hdfs\_server** server.

```
DROP SERVER hdfs_server;
```

### Helpful Links

[ALTER SERVER](#), [CREATE SERVER](#)

## 21.13.5 Views and Materialized Views

## 21.13.5.1 CREATE VIEW

### Function

**CREATE VIEW** creates a view.

A view is a virtual table, not a base table. A database only stores the definition of a view and does not store its data. The data is still stored in the original base table. If data in the base table changes, the data in the view changes accordingly. In this sense, a view is like a window through which users can know their interested data and data changes in the database.

### Precautions

None

### Syntax

```
CREATE [OR REPLACE] [TEMP | TEMPORARY] VIEW view_name [(column_name [, ...])]
[WITH ({view_option_name [= view_option_value]} [, ...])]
AS query;
```

#### NOTE

- You can use **WITH (security\_barriers)** to create a relatively secure view. This prevents attackers from printing hidden base table data by using the **RAISE** statement of low-cost functions.
- When the **view\_independent** GUC parameter is enabled, columns can be deleted from common views. Note that if a column-level constraint exists, the corresponding column cannot be deleted.
- Duplicate CTE names are not supported when the view decoupling function is enabled.

Example:

```
CREATE TABLE t1(a1 INT, b1 INT);
CREATE TABLE t2(a2 INT, b2 INT, c2 INT);
CREATE OR REPLACE VIEW v1 AS WITH tmp AS (SELECT * FROM t2) ,tmp1 AS (SELECT b2,c2
FROM tmp WHERE b2 = (WITH RECURSIVE tmp(aa, bb) AS (SELECT a1,b1 FROM t1) SELECT bb
FROM tmp WHERE aa = c2)) SELECT c2 FROM tmp1;
```

### Parameter Description

**Table 21-195** CREATE VIEW parameters

| Parameter        | Description                                                                    | Value Range                                                              |
|------------------|--------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| OR REPLACE       | If a view with the same name already exists, redefine the view.                | -                                                                        |
| TEMP   TEMPORARY | Creates a temporary view.                                                      | -                                                                        |
| view_name        | Specifies the name of a view to be created. It is optionally schema-qualified. | A string, which must comply with the <a href="#">naming convention</a> . |

| Parameter                                                     | Description                                                                                                                                                                                                                     | Value Range                                                                                                                                    |
|---------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| column_name                                                   | Specifies an optional list of names to be used for columns of the view. If not given, the column names are deduced from the query.                                                                                              | A string, which must comply with the <a href="#">naming convention</a> .                                                                       |
| WITH<br>( {view_option_name [= view_option_value]} [, ... ] ) | Specifies the optional parameters for a view.<br>Currently, the only parameter supported by <b>view_option_name</b> is <b>security_barrier</b> , which should be enabled when a view is intended to provide row-level security. | If security_barrier is of the boolean type, the options are as follows: <ul style="list-style-type: none"><li>• TRUE</li><li>• FALSE</li></ul> |
| query                                                         | A <b>SELECT</b> or <b>VALUES</b> statement which will provide the columns and rows of the view.                                                                                                                                 | -                                                                                                                                              |

## Examples

Create a view consisting of columns whose **spcname** is **pg\_default**:

```
CREATE VIEW myView AS
SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';
```

Run the following command to redefine the existing view **myView** and create a view consisting of columns whose **spcname** is **pg\_global**:

```
CREATE OR REPLACE VIEW myView AS
SELECT * FROM pg_tablespace WHERE spcname = 'pg_global';
```

Create a view consisting of rows with **age** smaller than 12:

```
DROP TABLE IF EXISTS customer;

CREATE TABLE customer
(id int, name varchar(20) , age int)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(id);

INSERT INTO customer VALUES (1,'lily',10),(2, 'lucy',12),(3,'lilei',15);

CREATE VIEW customer_details_view_v1 AS
SELECT * FROM customer
WHERE age < 12;
```

## Updatable Views

After the **enable\_view\_update** parameter is enabled, the **INSERT**, **UPDATE**, **DELETE**, and **MERGE INTO** statements can be used to update simple views. Only 8.1.2 or later supports updates using the **MERGE INTO** statement.

Views that meet all the following conditions can be updated:

- The FROM clause in the view definition contains only one common table, which cannot be a system table, foreign table, DFS table, delta table, TOAST table, or error table.
- The view contains updatable columns, which are simple references to the updatable columns of the base table.
- The view definition does not contain the WITH, DISTINCT, GROUP BY, ORDER BY, FOR UPDATE, FOR SHARE, HAVING, TABLESAMPLE, LIMIT or OFFSET clause.
- The view definition does not contain the UNION, INTERSECT, or EXCEPT operation.
- The selection list of the view definition does not contain aggregate functions, window functions, or functions that return collections.
- Ensure that there is no trigger whose trigger occasion is **INSTEAD OF** in the view for **INSERT**, **UPDATE**, and **DELETE** statements. For the **MERGE INTO** statement, neither the view nor the underlying table can have triggers.
- The view definition does not contain sublinks.
- The view definition does not contain functions whose attribute is **VOLATILE**. The values of such functions can be changed during a table scan.
- The view definition does not set an alias for the column where the distribution key of the table resides, or name a common column as the distribution key column.
- When the RETURNING clause is used in the view update operation, columns in the view definition only come from the base table.

If the definition of the updatable view contains a WHERE condition, the condition restricts the UPDATE and DELETE statements from modifying rows on the base table. If the WHERE condition is not met after the UPDATE statement is executed, the updated rows cannot be queried in the view. Similarly, If the WHERE condition is not met after the INSERT statement is executed, the inserted data cannot be queried in the view. To insert, update, or delete data in a view, you must have the corresponding permission on the view and tables.

## Helpful Links

[ALTER VIEW](#), [DROP VIEW](#)

### 21.13.5.2 ALTER VIEW

#### Function

**ALTER VIEW** modifies all auxiliary attributes of a view. (To modify the query definition of a view, use **CREATE OR REPLACE VIEW**.)

#### Precautions

- Only the view owner can run **ALTER VIEW**.
- To change a view's schema, you must also have the CREATE permission on the new schema.
- To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have CREATE privilege on the view's schema.

- An administrator can change the owner relationship of any view.

## Syntax

- Set the default value of the view column.  

```
ALTER VIEW [IF EXISTS] view_name
 ALTER [COLUMN] column_name SET DEFAULT expression;
```
- Remove the default value of the view column.  

```
ALTER VIEW [IF EXISTS] view_name
 ALTER [COLUMN] column_name DROP DEFAULT;
```
- Change the owner of a view.  

```
ALTER VIEW [IF EXISTS] view_name
 OWNER TO new_owner;
```
- Renames a view. The new view name can be prefixed with the schema name of the original view. The schema name cannot be changed at the same time.  

```
ALTER VIEW [IF EXISTS] view_name
 RENAME TO new_name;
ALTER VIEW [IF EXISTS] view_name
 RENAME TO schema.new_name;
```
- Set the schema of the view.  

```
ALTER VIEW [IF EXISTS] view_name
 SET SCHEMA new_schema;
```
- Set the options of the view.  

```
ALTER VIEW [IF EXISTS] view_name
 SET ({ view_option_name [= view_option_value] } [, ...]);
```
- Reset the options of the view.  

```
ALTER VIEW [IF EXISTS] view_name
 RESET (view_option_name [, ...]);
```
- Rebuild the current view and its lower-layer and upper-layer dependent views.  

```
ALTER VIEW [IF EXISTS] view_name
 REBUILD;
```
- Rebuild the current view and its lower-layer dependent views.  

```
ALTER VIEW [IF EXISTS] ONLY view_name
 REBUILD;
```

## Parameter Description

**Table 21-196** ALTER VIEW parameters

| Parameter   | Description                                                                                                                        | Value Range        |
|-------------|------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| IF EXISTS   | Sends a prompt instead of an error if the view does not exist.                                                                     | -                  |
| view_name   | Specifies the name of the view to be modified, which can include a schema.                                                         | Valid view name.   |
| column_name | Indicates an optional list of names to be used for columns of the view. If not given, the column names are deduced from the query. | Valid column name. |

| Parameter                              | Description                                                                                                                                                                                                                                                                                                         | Value Range                                                                                                                                 |
|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| SET/DROP<br>DEFAULT                    | Sets or deletes the default value of a column. Currently, this parameter does not take effect.                                                                                                                                                                                                                      | -                                                                                                                                           |
| new_owner                              | Specifies the new owner of the view.                                                                                                                                                                                                                                                                                | Valid username.                                                                                                                             |
| new_name                               | Specifies the new view name.                                                                                                                                                                                                                                                                                        | A string compliant with the <a href="#">identifier naming rules</a> .                                                                       |
| new_schema                             | Specifies the new schema of the view.                                                                                                                                                                                                                                                                               | Valid schema name.                                                                                                                          |
| view_option_name [= view_option_value] | This clause specifies optional parameters for a view.<br>Currently, the only parameter supported by <b>view_option_name</b> is <b>security_barrier</b> , which should be enabled when a view is intended to provide row-level security.                                                                             | <b>security_barrier</b> is of the boolean type. The options include: <ul style="list-style-type: none"><li>• TRUE</li><li>• FALSE</li></ul> |
| REBUILD                                | This clause is used for view decoupling. You can use the saved original statement to rebuild views and restore the dependencies. In 9.1.0.200 and later cluster versions, the view decoupling function does not require <b>view_independent</b> to be enabled. If a view becomes invalid, it can simply be rebuilt. | For how to use <b>REBUILD</b> , see <a href="#">REBUILD Usage Description</a> .                                                             |
| ONLY                                   | Limits view rebuilding to only the selected view and its dependencies. This function does not require <b>view_independent</b> to be enabled. If a view becomes invalid, it can simply be rebuilt.                                                                                                                   | -                                                                                                                                           |

## REBUILD Usage Description

- View rebuilding starts from the current view and updates all associated backward views. If the forward views on which the current view depends are also unavailable, automatic rebuilding is triggered.
- The temporary tables and views that have dependency relationships cannot be decoupled and dropped. However, you can perform the REBUILD operation on temporary views that do not have dependency relationships.

- View schema names and view names can be modified. The names of rebuilt view schemas or views are re-created based on the latest name, but the query operation retains the original definition.
- Only fields of the character, number, and time types in the base table can be modified. Only fields of the character, number, and time types in the base table can be modified. When a field is added to the base table, the view is not invalidated and the definition remains unchanged.
- Invalid views are exported as comments during backup. You need to manually restore the invalid views.

 **NOTE**

The upper-layer cascading views become invalid in the following scenarios:

- DROP TABLE/VIEW
- RENAME TABLE/VIEW
- ALTER TABLE DROP COLUMN
- ALTER TABLE CHANGE/ALTER COLUMN TYPE
- ALTER TABLE CHANGE/ALTER COLUMN NAME
- ALTER TABLE/VIEW NAMESPACE
- ALTER TABLE/VIEW RENAME

## Examples

Create the sample view **myview**.

```
CREATE OR REPLACE VIEW myview AS
SELECT * FROM pg_tablespace WHERE spcname = 'pg_default';
```

Change the view name.

```
ALTER VIEW myview RENAME TO product_view;
```

Change the schema of a view.

```
ALTER VIEW product_view SET schema public;
```

Rebuild a view.

```
ALTER VIEW public.product_view REBUILD;
```

Rebuild a dependent view.

```
ALTER VIEW ONLY public.product_view REBUILD;
```

## Helpful Links

[CREATE VIEW](#) and [DROP VIEW](#)

### 21.13.5.3 DROP VIEW

#### Function

**DROP VIEW** forcibly deletes an existing view in a database.

#### Precautions

Only a view owner or a system administrator can run **DROP VIEW** command.



**⚠ DANGER**

- Be cautious when using **DROP OBJECT** (e.g., **DATABASE, USER/ROLE, SCHEMA, TABLE, VIEW**) as it may cause data loss, especially with **CASCADE** deletions. Always back up data before proceeding.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

## Syntax

```
DROP VIEW [IF EXISTS] view_name [, ...] [CASCADE | RESTRICT];
```

## Parameter Description

**Table 21-197** DROP VIEW parameters

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                                          |
|--------------------|-----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF EXISTS          | Sends a notice instead of an error if the specified view does not exist.                            | -                                                                                                                                                                                                                                                    |
| view_name          | Specifies the name of the view to be deleted.                                                       | An existing view.                                                                                                                                                                                                                                    |
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: deletes objects (such as other views) that depend on a view to be deleted.</li><li>• <b>RESTRICT</b>: refuses to delete the view if any objects depend on it. This is the default.</li></ul> |

## Examples

Delete **myView** view.

```
DROP VIEW myView;
```

Delete **customer\_details\_view\_v2** view.

```
DROP VIEW public.customer_details_view_v2;
```

## Helpful Links

[ALTER VIEW](#), [CREATE VIEW](#)

## 21.13.5.4 CREATE MATERIALIZED VIEW

### Function

CREATE MATERIALIZED VIEW creates a materialized view. This syntax is supported only by clusters of version 8.2.1.220 or later.

A materialized view is a special database object. It pre-computes complex query results and stores them in the database to accelerate queries.

When a large amount of data is frequently queried and analyzed, materialized views can be used to pre-calculate and store query results, significantly improving query performance.

### Materialized View Permissions

1. To create a materialized view, you must have the **CREATE** permission on the schema and the **SELECT** permission on the base table or column.
2. To query a materialized view, you must have the **SELECT** permission on the materialized view.
3. To refresh a materialized view, you must have the **INSERT** permission on the view and the **SELECT** permission on the base table or column.
4. Other permissions:

Materialized views support fine-grained permissions such as **ANALYZE**, **VACUUM**, **ALTER**, and **DROP**.

Materialized views support **WITH GRANT OPTION**, which means that the recipient of the permission can in turn grant it to others.

5. Security:

Materialized views do not support higher-level security control. In scenarios where the **SELECT** permission is restricted, for example, a base table has row-level access control or masking policies or its owner is a private user, materialized views cannot be created.

If a materialized view already exists and an RLS or masking policy is added to the base table or the owner of the base table is changed to a private user, the materialized view can be queried but cannot be refreshed.

### Precautions

- To use **CREATE MATERIALIZED VIEW**, ensure that **enable\_matview** is set to **on**.
- The base table of a materialized view can be a row-store table, column-store table, HStore table, hot and cold table (supported by clusters of version 9.1.0.200 or later), partitioned table, specified partition, foreign table, or another materialized view.
- Temporary tables (including global temporary tables, volatile temporary tables, and common temporary tables) are not supported. Partitions cannot be specified for partitioned tables that are automatically created.
- The CTE in the definition of a materialized view cannot contain other DML statements except the **SELECT** statement.
- The base table of a materialized view cannot have masking policies and row-level access control, or belong to a private user. Masking policies and row-

level access control may cause result set errors, while private users may cause problems such as materialized view refresh failures.

- Materialized views do not support the following storage types: foreign tables and time series tables.
- Materialized views do not allow **INSERT**, **UPDATE**, **MERGE INTO**, or **DELETE** operations.
- The result of each query to the materialized view saved to ensure that the query results are the same each time. After the **BUILD IMMEDIATE** or **REFRESH** operation, the correct result can be queried in the materialized view.
- A node group cannot be specified for a materialized view using syntax. A node group can be specified for the base table of a materialized view. The materialized view can inherit the node group information from the base table. The node groups of multiple base tables must be the same.

## Syntax

```
CREATE MATERIALIZED VIEW [IF NOT EXISTS] materialized_view_name [(column_name [, ...])]
[BUILD { DEFERRED | IMMEDIATE }]
[REFRESH [[DEFAULT] | [RESTRICT] | [CASCADE FORWARD] | [CASCADE BACKWARD] | [CASCADE
ALL]] [COMPLETE] [ON DEMAND] [[START WITH (timestampz)] | [EVERY (interval)]]]
[{ ENABLE | DISABLE } QUERY REWRITE]
[WITH ({storage_parameter = value} [, ...])]
[DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH (column_name [,...]) } }]
[PARTITION BY {(column_name) | date_trunc('time_unit', column_name)}]
AS query;
```

## Parameter Description

**Table 21-198** CREATE MATERIALIZED VIEW parameters

| Parameter                     | Description                                                                                                                                                                                                                | Value Range/Default Value |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| BUILD DEFERRED<br>  IMMEDIATE | <b>IMMEDIATE</b> indicates that the latest data is included when the materialized view is created.<br><br><b>DEFERRED</b> indicates that data is included only when the materialized view is refreshed for the first time. | -                         |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                  | Value Range/Default Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| REFRESH   | <p>Specifies the refresh mode of a materialized view.</p> <p>After a materialized view is created, the data in the materialized view reflects only the status of the base table at the creation time. When the data in the base table changes, you need to refresh the materialized view (<b>REFRESH MATERIALIZED VIEW</b>) to update the data in the materialized view.</p> | <ul style="list-style-type: none"><li>Refresh method: The options include <b>DEFAULT</b> (default refresh) and <b>RESTRICT</b> (forcible refresh). You can only specify the refresh method for materialized views that are asynchronously refreshed in the background.</li><li>Refresh direction: When using materialized views in a cascading nested manner, the refresh direction can be specified as <b>FORWARD</b>, <b>BACKWARD</b>, or <b>ALL</b>. You can only specify the refresh direction for materialized views that are asynchronously refreshed in the background. For details, see <a href="#">REFRESH MATERIALIZED VIEW</a>.</li><li>Currently, only the <b>COMPLETE</b> refresh mode is supported, which refresh full data in the materialized view. Execute the query statement defined in the materialized view to update the materialized view.</li><li>Refresh triggering mode. <b>ON DEMAND</b>: manual refresh on demand.<br/><b>START WITH (timestamp tz)   EVERY (interval)</b>: automatic refresh at a scheduled time. <b>START WITH</b> specifies the first refresh time. <b>EVERY</b> specifies the automatic refresh interval and its value can be <b>MONTH</b>, <b>DAY</b>, <b>hour</b>, <b>MINUTE</b>, or <b>SECOND</b>.</li></ul> |

| Parameter                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                               | Value Range/Default Value                                                                                                                                                                       |
|-----------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ENABLE   DISABLE<br>QUERY REWRITE | <p>Specifies whether to enable query rewrite.</p> <p>Query rewrite means that when a base table is queried and a materialized view is created for the base table, the database system automatically determines whether the pre-calculated result in the materialized view can be used to process the query.</p> <p>If a materialized view can be used, the pre-calculated result is directly read from the materialized view to accelerate the query.</p> | <p>Disabled by default.</p> <p>When <b>ENABLE QUERY REWRITE</b> is specified, you need to configure the GUC parameter <b>mv_rewrite_rule</b> to enable query rewrite of materialized views.</p> |
| WITH                              | Specifies the parameters for creating a materialized view.                                                                                                                                                                                                                                                                                                                                                                                                | For details, see <a href="#">Table 21-199</a> .                                                                                                                                                 |

| Parameter     | Description                                                       | Value Range/Default Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------|-------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DISTRIBUTE BY | Specifies how the table is distributed or replicated between DNs. | <p>Value range: <b>REPLICATION</b>, <b>ROUNDROBIN</b>, or <b>HASH</b></p> <ul style="list-style-type: none"><li>• <b>REPLICATION</b>: Each row in the table exists on all DNs, that is, each DN has complete table data.</li><li>• <b>ROUNDROBIN</b>: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. This value is supported only in 8.1.2 or later.</li><li>• <b>HASH</b>: Each row of the table will be placed into all the DNs based on the hash value of the specified column.</li></ul> <p>Default value: determined by the <b>default_distribution_mode</b> parameter.</p> <p><b>NOTE</b></p> <p>When the materialized view is distributed in hash mode, data skew may occur. To check for data skews in materialized views, follow the same procedures used for detecting data skews in regular tables. For details, see "Checking for Data Skew" in the <i>Data Warehouse Service (DWS) Developer Guide</i>. If data skew is identified in a materialized view, perform data skew optimization at the storage layer by referring to "Optimizing Data Skew" in the <i>Data Warehouse Service (DWS) Developer Guide</i>.</p> |

| Parameter    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Value Range/Default Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PARTITION BY | <p>Creates a partitioned materialized view. This parameter is supported only by clusters of version 9.1.0.200 or later.</p> <p><b>CAUTION</b></p> <ul style="list-style-type: none"><li>• The partition key of a materialized view must match that of the queried table's partitioned table to ensure partition-based mapping for invalidation and refresh.</li><li>• The partition key of a materialized view must be included in the outermost SQL statement of the query to enable partition key-based updates to the materialized view.</li><li>• Materialized views support only range partitioning.</li><li>• The queried table can be a composite partition. If multiple partition tables have the same partition conditions, the mapping between the queried table and the materialized view is automatically established.</li><li>• Partition mapping of foreign tables is not supported because data changes in foreign tables cannot be detected.</li><li>• Columns that generate null values in outer joins cannot be used as partition keys.</li><li>• Merging and splitting of base table partitions are not supported.</li></ul> | <ul style="list-style-type: none"><li>• Proportional association: Specify the partition key of a partitioned table in the query as the partition key of the materialized view, creating a one-on-one mapping relationship.</li><li>• Roll-up association: Use the <b>date_trunc</b> function to roll up the partition key of a partitioned table in the query, establishing a roll-up mapping relationship. The data type should be compatible with the <b>date_trunc</b> function.</li></ul> |
| AS query     | Creates a materialized view based on the query result.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

Table 21-199 WITH parameters

| Parameter                          | Description                                                                                                                                                                                                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ORIENTATION                        | Specifies the storage mode (row-store, column-store) for table data. This parameter cannot be modified once it is set.                                                                                                                                                                                                                                            | <b>ROW</b> or <b>COLUMN</b> <ul style="list-style-type: none"><li>• <b>ROW</b> indicates that table data is stored in rows. <b>ROW</b> applies to OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using ROW can improve the efficiency.</li><li>• <b>COLUMN</b> indicates that the data is stored in columns. Column storage applies to the data warehouse service, which has a large amount of aggregation computing, and involves a few column operations.</li></ul> Default value: <b>ROW</b> |
| enable_foreign_table_query_rewrite | Specifies whether to allow query rewriting on materialized views that contain foreign tables. This parameter must be used together with <b>ENABLE QUERY REWRITE</b> .<br>The materialized view cannot detect the data changes in the foreign table. Specify this option if you want to enable query rewriting for materialized views that contain foreign tables. | <ul style="list-style-type: none"><li>• <b>on</b>: allows query rewriting on materialized views that contain foreign tables.</li><li>• <b>off</b>: does not allow query rewriting on materialized views that contain foreign tables.</li></ul> Default value: <b>off</b>                                                                                                                                                                                                                                                                                   |
| bitmap_columns                     | The bitmap index is only applicable to the <b>hstore_opt</b> table. To use it, enable the table-level parameter <b>enable_hstore_opt</b> and set <b>bitmap_columns</b> to the specified column. This is supported only by clusters of version 9.1.0.200 or later.                                                                                                 | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |



| Parameter          | Description                                                                                                                                                                                                                    | Value Range                                                                                                                                                                                                                                                                                                      |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| secondary_part_num | Specifies the number of level-2 partitions in a column-store table. This parameter applies only to H-Store column-store tables. This is supported only by clusters of version 9.1.0.200 or later.                              | Value range: <b>1</b> to <b>32</b><br>Default value: <b>8</b>                                                                                                                                                                                                                                                    |
| enable_hstore_opt  | When the <b>enable_hstore_opt</b> table-level parameter is enabled, the <b>enable_hstore</b> table-level parameter is also automatically enabled by default. This is supported only by clusters of version 9.1.0.200 or later. | <b>true</b> or <b>false</b><br>Default value: <b>false</b>                                                                                                                                                                                                                                                       |
| enable_turbo_store | Determines whether to create a turbo table (column-store tables). The parameter is only valid for column-store tables. This is supported only by clusters of version 9.1.0.200 or later.                                       | <b>on</b> or <b>off</b><br>Default value: <b>off</b>                                                                                                                                                                                                                                                             |
| mv_analyze_mode    | Determines the automatic analysis method for materialized views. This is supported only by clusters of version 9.1.0.200 or later.                                                                                             | <ul style="list-style-type: none"><li>• <b>none</b>: indicates that the materialized view does not automatically execute <b>ANALYZE</b> after being refreshed.</li><li>• <b>light</b>: indicates that the materialized view performs light analysis after being refreshed.</li></ul> Default value: <b>light</b> |

| Parameter                | Description                                                                                                                                                                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                         |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mv_pck_column            | Specifies a partial clustering storage for a materialized view. During the data import process to a column-store table, the data is partially sorted according to the specified column(s). This is supported only by clusters of version 9.1.0.200 or later.<br><br>Enable <b>mv_pck_column</b> and set it to a specified column. | -                                                                                                                                                                                                                                                                                                                   |
| mv_support_function_type | Enables the use of function attributes in the query statements when creating materialized views. This is supported only by clusters of version 9.1.0.200 or later.                                                                                                                                                                | <ul style="list-style-type: none"><li>• <b>stable</b>: indicates that functions of the STABLE and IMMUTABLE types can be used in query statements.</li><li>• <b>volatile</b>: indicates that functions of the VOLATILE, STABLE, and IMMUTABLE types can be used in query statements.</li></ul> Default value: empty |
| excluded_inactive_tables | Specifies that the materialized view will not be invalidated when there are data changes in the underlying base table. This is supported only by clusters of version 9.1.0.200 or later.<br><br>Set <b>excluded_inactive_tables</b> to <b>schemaName1.tableName1,schemaName2.tableName2</b> .                                     | Default value: empty                                                                                                                                                                                                                                                                                                |
| force_rewrite_timeout    | Allows query rewrite within a specified time interval after refresh, regardless of the freshness of the data in the materialized view. This is supported only by clusters of version 9.1.0.200 or later.                                                                                                                          | The unit is second. The default value is <b>0</b> .                                                                                                                                                                                                                                                                 |

| Parameter                     | Description                                                                                                                                                                                     | Value Range |
|-------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| TABLESPACE<br>tablespace_name | Specifies a tablespace for V3 storage. If <b>default_tablespace</b> is empty, the default tablespace of the database is used. This is supported only by clusters of version 9.1.0.200 or later. | -           |

## Examples

Create a base table and insert data into the base table.

```
CREATE TABLE t1 (a int, b int) DISTRIBUTE BY HASH(a);
INSERT INTO t1 SELECT x,x FROM generate_series(1,10) x;
```

Create a materialized view with the default option **BUILD IMMEDIATE**.

```
CREATE MATERIALIZED VIEW mv1 AS SELECT * FROM t1;
```

Create a materialized view in column-store mode.

```
CREATE MATERIALIZED VIEW mv2 WITH(orientation = column) AS SELECT * FROM t1;
```

Create a materialized view that is manually refreshed as required.

```
CREATE MATERIALIZED VIEW mv3 BUILD DEFERRED REFRESH ON DEMAND AS SELECT * FROM t1;
```

Create a materialized view with a scheduled refresh time.

```
CREATE MATERIALIZED VIEW mv4 BUILD DEFERRED REFRESH START WITH(trunc(sysdate)) EVERY
(interval '1 day') AS SELECT * FROM t1;
```

Create a materialized view with a bitmap index.

```
CREATE MATERIALIZED VIEW mv1
with (ORIENTATION = COLUMN, enable_hstore=true, enable_hstore_opt=on, bitmap_columns='col1') AS
SELECT * FROM base_table;
```

Create a materialized view and specify the number of level-2 partitions in a column-store table.

```
CREATE MATERIALIZED VIEW mv
WITH (ORIENTATION=COLUMN, ENABLE_HSTORE=ON, enable_hstore_opt=on, mv_pck_column='c3',
secondary_part_column = 'c2', secondary_part_num = 8) AS SELECT * FROM base_table;
```

Create a materialized view and specify the PCK column for sorting.

```
CREATE MATERIALIZED VIEW mv
WITH (ORIENTATION=COLUMN, ENABLE_HSTORE=ON, enable_hstore_opt=on, mv_pck_column='col3') AS
SELECT * FROM base_table;
```

Create a materialized view and specify the analysis method.

```
CREATE MATERIALIZED VIEW mv1 enable query rewrite
with(excluded_inactive_tables='matview_basic."T1",matview_basic."a=b"',mv_analyze_mode='none') as
SELECT * FROM base_table;
```

Create a V3 materialized view.

```
CREATE MATERIALIZED VIEW mv1
with (orientation=column, enable_hstore=true, compression=low, enable_hstore_opt=on, COLVERSION =
3.0) TABLESPACE cu_obs_tbs distribute by hash(scope_name)
AS SELECT * FROM dicttbl_low;
```

Create a materialized view that contains a foreign table for query rewriting.

```
CREATE MATERIALIZED VIEW mv1 with (enable_foreign_table_query_rewrite = true) as SELECT * FROM
base_table;
```

Create a materialized view and specify that volatile functions can be used in the query statement.

```
CREATE MATERIALIZED VIEW mv_date with(mv_support_function_type = 'volatile') as select to_date(a)
from t_date;
```

## Helpful Links

[ALTER MATERIALIZED VIEW](#), [DROP MATERIALIZED VIEW](#), [REFRESH MATERIALIZED VIEW](#)

### 21.13.5.5 ALTER MATERIALIZED VIEW

#### Function

Modifies the properties of a materialized view. This syntax is supported only by clusters of version 8.2.1.220 or later.

#### Precautions

To use ALTER MATERIALIZED VIEW, ensure that **enable\_matview** is set to **on**.

#### Syntax

- Set whether to enable query rewrite for materialized views.  

```
ALTER MATERIALIZED VIEW [IF EXISTS] { materialized_view_name }
[ENABLE | DISABLE] QUERY REWRITE;
```
- Specify how materialized views are refreshed. Currently, only **COMPLETE** is available, which rebuilds the entire materialized view from scratch, re-executing the defining query.  

```
ALTER MATERIALIZED VIEW [IF EXISTS] { materialized_view_name }
REFRESH [[DEFAULT]] [[RESTRICT]] [[CASCADE FORWARD]] [[CASCADE BACKWARD]]
[[CASCADE ALL]] [[COMPLETE]] [[ON DEMAND]] [[START WITH (timestamp tz)]] [[EVERY
(interval)]];
```

  - REFRESH ON DEMAND** indicates that the data is manually refreshed as required.
  - START WITH** specifies the first refresh time.
  - EVERY** specifies the refresh interval. The value can be **MONTH**, **DAY**, **HOURL**, **MINUTE**, or **SECOND**.
- Change how materialized views refresh asynchronously in the background. This change applies only to materialized views that refresh this way. This syntax is supported only by clusters of version 9.1.0.200 or later. For details, see [REFRESH MATERIALIZED VIEW](#).  

```
ALTER MATERIALIZED VIEW [IF EXISTS] { materialized_view_name }
REFRESH [DEFAULT] [[RESTRICT]] [[CASCADE FORWARD]] [[CASCADE BACKWARD]]
[[CASCADE ALL]];
```

- Change the owner of the materialized view.  

```
ALTER MATERIALIZED VIEW { materialized_view_name }
OWNER TO new_owner;
```
- Allow for setting table properties of materialized views. This syntax is supported only by clusters of 9.1.0.200 and later versions.  

```
ALTER MATERIALIZED VIEW { materialized_view_name }
SET ({storage_parameter = value} [, ...])
| RESET (storage_parameter [, ...])
```
- Rename a view. Materialized views cannot be renamed across schemas. When renaming a materialized view, you can specify a schema name for **materialized\_view\_name** but cannot specify a schema name for **new\_materialized\_view\_name**.  

```
ALTER MATERIALIZED VIEW [IF EXISTS] materialized_view_name
RENAME TO new_materialized_view_name;
```

## Parameter Description

Table 21-200 ALTER MATERIALIZED VIEW parameters

| Parameter                         | Description                                                                                   | Value Range                                                                                                                                                                          |
|-----------------------------------|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF EXISTS                         | If the specified materialized view does not exist, a message instead of an error is returned. | -                                                                                                                                                                                    |
| materialized_view_name            | Specifies the name of the materialized view to be modified.                                   | Name of an existing materialized view.                                                                                                                                               |
| ENABLE   DISABLE<br>QUERY REWRITE | Specifies whether to enable query rewrite.                                                    | Disabled by default.<br>When <b>ENABLE QUERY REWRITE</b> is specified, you need to configure the GUC parameter <b>mv_rewrite_rule</b> to enable query rewrite of materialized views. |

| Parameter | Description                                                                                                                                                                                        | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| REFRESH   | <p>Specifies how materialized views are refreshed.</p> <p>When the data in base tables changes, you need to run <b>REFRESH MATERIALIZED VIEW</b> to update the data in the materialized views.</p> | <ul style="list-style-type: none"><li>Refresh method: <b>DEFAULT</b> (default refresh) and <b>RESTRICT</b> (forcible refresh). You can specify the refresh method only for materialized views that are asynchronously refreshed in the background.</li><li>Refresh direction: When using materialized views in a cascading nested manner, the refresh direction can be specified as <b>FORWARD</b>, <b>BACKWARD</b>, or <b>ALL</b>. You can specify the refresh direction only for materialized views that are asynchronously refreshed in the background. For details, see <a href="#">REFRESH MATERIALIZED VIEW</a>.</li><li>Currently, only <b>COMPLETE</b> is available, which rebuilds the entire materialized view from scratch, re-executing the defining query.</li><li>How to trigger a refresh:<br/><b>ON DEMAND</b>: manual refresh on demand.<br/><b>START WITH (timestamp tz)   EVERY (interval)</b>: scheduled refresh.<br/><b>START WITH</b> specifies the first refresh time.<br/><b>EVERY</b> specifies the refresh interval and its value can be <b>MONTH</b>, <b>DAY</b>, <b>HOUR</b>, <b>MINUTE</b>, or <b>SECOND</b>.</li></ul> |

| Parameter                  | Description                                                                                                                                                                                                                                                                             | Value Range                                                           |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| storage_parameter          | Allows for setting table properties of materialized views.<br><br>Parameters such as <b>mv_pck_column</b> , <b>bitmap_columns</b> , <b>enable_foreign_table_query_rewrite</b> , <b>excluded_inactive_tables</b> , <b>force_rewrite_timeout</b> , and <b>mv_analyze_mode</b> can be set. | For details, see <a href="#">Parameter Description</a> .              |
| new_owner                  | Specifies the owner of a materialized view.                                                                                                                                                                                                                                             | Name of an existing user.                                             |
| new_materialized_view_name | Indicates the name of the new materialized view.                                                                                                                                                                                                                                        | A string compliant with the <a href="#">identifier naming rules</a> . |

## Examples

Enable query rewrite for a materialized view.

```
ALTER MATERIALIZED VIEW mv1 ENABLE QUERY REWRITE;
NOTICE: REFRESH MATERIALIZED VIEW should be executed to enable query rewrite.
ALTER MATERIALIZED VIEW
```

Modify the table properties of a materialized view.

```
ALTER MATERIALIZED VIEW mv1 SET (force_rewrite_timeout=100);
ALTER MATERIALIZED VIEW mv1 SET (mv_pck_column='col1');
ALTER MATERIALIZED VIEW mv1 SET (enable_foreign_table_query_rewrite = true);
```

Change the refresh time of the materialized view.

```
ALTER MATERIALIZED VIEW mv1 REFRESH START WITH('2025-01-01 15:15:15'::timestampz) EVERY
(interval '60 s');
```

## Helpful Links

[CREATE MATERIALIZED VIEW](#), [DROP MATERIALIZED VIEW](#), [REFRESH MATERIALIZED VIEW](#)

### 21.13.5.6 REFRESH MATERIALIZED VIEW

#### Function

REFRESH MATERIALIZED VIEW refreshes materialized views. This syntax is supported only by clusters of version 8.2.1.220 or later.

The refresh mode is specified by the REFRESH parameter in the [CREATE MATERIALIZED VIEW](#) syntax. Currently, full refresh and scheduled refresh are supported.

## Precautions

The refresh operation blocks the DML operations on the base table.

## Syntax

```
REFRESH MATERIALIZED VIEW
{[schema.]materialized_view_name} [[DEFAULT] | [RESTRICT] | [CASCADE FORWARD] | [CASCADE
BACKWARD] | [CASCADE ALL]]
```

## Parameter Description

**Table 21-201** REFRESH MATERIALIZED VIEW parameters

| Parameter                      | Description                                                                                                                                                                                                                                                                                                                                               | Value Range |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| materialize<br>d_view_na<br>me | Specifies the name of the materialized view to be refreshed.                                                                                                                                                                                                                                                                                              | -           |
| DEFAULT                        | Specifies default refresh. It means that a single materialized view is not refreshed only until the base table has changed.                                                                                                                                                                                                                               | -           |
| RESTRICT                       | Specifies forced refresh. It means that a materialized view is refreshed regardless of whether the base table has changed.                                                                                                                                                                                                                                | -           |
| CASCADE<br>FORWARD             | Indicates a cascading refresh in the upward direction, starts by refreshing the current materialized view, and then refreshes the nested materialized views that depend it. The default refresh method is used.                                                                                                                                           | -           |
| CASCADE<br>BACKWARD            | Specifies cascading refresh in the downward direction. The system refreshes the lowest-level materialized view that the current materialized view relies on, then continues to refresh the nested materialized views that depend on the lowest-level one, and finally refreshes the current materialized view itself. The default refresh method is used. | -           |
| CASCADE<br>ALL                 | Specifies all cascading refreshes. First, cascading refreshes are performed downward, and then cascading refreshes are performed upward.                                                                                                                                                                                                                  | -           |



## Examples

Refresh a materialized view.

```
REFRESH MATERIALIZED VIEW mv1;
```

Perform a cascading refresh in the downward direction for materialized views.

```
CREATE TABLE t1(a int,b int);
CREATE MATERIALIZED VIEW mv1 AS SELECT * FROM t1 where a > 0;
CREATE MATERIALIZED VIEW mv2 AS SELECT * FROM mv1;
INSERT INTO t1 values(1,1),(2,2);
REFRESH MATERIALIZED VIEW mv2 cascade backward; -- After refreshing mv2, the data in mv1 was also updated.

SELECT * FROM mv1 ORDER BY 1,2;
a | b
---+---
1 | 1
2 | 2
(2 rows)

SELECT * FROM mv2 ORDER BY 1,2;
a | b
---+---
1 | 1
2 | 2
(2 rows)
```

## Helpful Links

[CREATE MATERIALIZED VIEW](#), [ALTER MATERIALIZED VIEW](#), [DROP MATERIALIZED VIEW](#)

### 21.13.5.7 DROP MATERIALIZED VIEW

#### Function

Deletes a materialized view. This syntax is supported only by clusters of version 8.2.1.220 or later.

#### Precautions

None

#### Syntax

```
DROP MATERIALIZED VIEW [IF EXISTS]
{[schema.]materialized_view_name} [, ...] [CASCADE | RESTRICT];
```

#### Parameter Description

**Table 21-202** DROP MATERIALIZED VIEW parameters

| Parameter | Description                                                                           | Value Range |
|-----------|---------------------------------------------------------------------------------------|-------------|
| IF EXISTS | Sends a notice instead of an error if the specified materialized view does not exist. | -           |

| Parameter                  | Description                                                                                         | Value Range                                                                                                                                                                                                                                                 |
|----------------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| materialize<br>d_view_name | Name of the materialized view to be deleted.                                                        | Name of an existing materialized view.                                                                                                                                                                                                                      |
| CASCADE  <br>RESTRICT      | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: automatically deletes all objects that are contained in the schema to be deleted.</li><li>• <b>RESTRICT</b>: refuses to delete the schema that contains any objects. This is the default.</li></ul> |

## Examples

Delete a materialized view.

```
DROP MATERIALIZED VIEW mv1;
```

## Helpful Links

[ALTER MATERIALIZED VIEW](#), [CREATE MATERIALIZED VIEW](#), [REFRESH MATERIALIZED VIEW](#)

## 21.13.6 Indexes, Sequences, and File Directories

### 21.13.6.1 CREATE INDEX

#### Function

**CREATE INDEX** creates an index in a specified table. An index in a database is a specialized data structure designed to significantly drive data retrieval operations. An index acts like a sorted directory, allowing the database system to quickly locate specific rows without needing to scan the entire table.

An index has the following functions:

1. **Accelerating queries**

In database applications, full table scans on large datasets cause slow queries. Creating indexes resolves this by allowing the database to locate data directly, bypassing the need to scan every row. For example, an index on an **id** column lets the system instantly find the record **where id=100**, dramatically improving performance.

2. **Optimizing sorting and grouping operations**

Indexes such as B-tree indexes are ordered. When a query involves **ORDER BY** or **GROUP BY**, you can directly use the order of indexes to avoid extra sorting operations and improve efficiency.

### 3. Restricting data integrity

Some indexes (such as primary key indexes and unique indexes) can enforce unique values of columns (for example, primary keys cannot be duplicate), helping maintain data integrity.

## Index Types

DWS supports the following types of indexes:

- B-tree indexes store data keys using a structure like the B+ tree, enabling fast index searches. B-tree indexes support range queries, sorting, and equality queries.
- GIN indexes are inverted index designed for complex data types that contain multiple searchable keys, such as arrays. It is ideally suited for multi-value matching, full-text search, and fuzzy queries.
- GiST indexes apply to multi-dimensional data types (such as geometry and geography) and set data types. They apply to multi-dimensional data, spatial search, and complex scopes.
- PSORT indexes are used to perform partial sort on column-store tables.

Notes:

- Row-store tables: B-tree (default), GIN, and GiST indexes
- Column-store tables: PSORT (default), B-tree, and GIN indexes

Partitioned tables have two index types: local and global.

- Local indexes (partitioned indexes) apply to individual partitions within a partitioned table. Each local index matches one partition and shares its count and range with the table. If a table partition changes, only that specific index partition requires maintenance.
- Global indexes span across all partitions, covering the entire table's data regardless of how the base table is partitioned. However, DWS currently does not support global indexes.

## Indexing Methods

**Table 21-203** Indexing methods

| Method            | Description                                                                                                                                                                                                                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unique indexes    | An index ensures the uniqueness of one or multiple attributes by acting as a constraint. When a table defines a unique constraint or primary key, DWS generates a unique index—possibly for multiple columns—on those specific columns to enforce this rule. In DWS, only B-tree supports creating unique indexes. |
| Composite indexes | A table's index can include multiple attributes. In DWS, B-tree allows multi-column indexing with up to 32 columns.                                                                                                                                                                                                |

| Method           | Description                                                                                                                                                                               |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Partial indexes  | Refers to an index that can be created for subsets of a table. This indexing method contains only tuples that meet condition expressions.                                                 |
| Expression index | An index can be built on a function or expression derived from one or more table columns. This type of index functions only if queries use the exact expression used during its creation. |

## Principles for Creating Indexes

Indexes are primarily used to enhance database performance (though inappropriate use can result in slower database performance). You are advised to create indexes on:

- Columns that are often queried
- Join conditions. For a query on joined columns, you are advised to create a composite index on the columns, for example, **select \* from t1 join t2 on t1.a=t2.a and t1.b=t2.b**. You can create a composite index on the **a** and **b** columns of table **t1**.
- Columns having filter criteria (especially scope criteria) of a **where** clause
- Columns that appear after **order by**, **group by**, and **distinct**.
- For point queries, create a B-tree index.

The index creation syntax on a partitioned table differs from that on an ordinary table. The partitioned table does not support concurrent index creation, partial index creation, or **NULL FIRST**.

## Precautions

- Indexes consume storage and computing resources. Creating too many indexes has negative impact on database performance (especially the performance of data import. Therefore, you are advised to import the data before creating indexes). Create indexes only when they are necessary.
- All functions and operators used in an index definition must be immutable, that is, their results must depend only on their arguments and never on any outside influence (such as the contents of another table or the current time). This restriction ensures that the behavior of the index is well-defined. When using a user-defined function on an index or in a **WHERE** statement, mark it as an immutable function.
- An index created on a partitioned table must include a partition column and all the partition keys.
- DWS can only create local indexes on partitioned tables.
- Column-store and HDFS tables support B-tree indexes. If the indexes are used, you cannot create expression and partial indexes.
- Column-store tables support creating unique indexes using B-tree indexes.
- Column-store and HDFS tables support PSORT indexes. If the PSORT indexes are used, you cannot create expression, partial, or unique indexes.

- Column-store tables support GIN indexes, rather than partial indexes and unique indexes. If GIN indexes are used, you can create expression indexes. However, an expression in this situation cannot contain empty splitters, empty columns, or multiple columns.
- Column-store indexes do not support **OR** query filter criteria and **inlist** operations.
- A primary key or unique index cannot be created for a round-robin table.
- Performing **CREATE INDEX** or **REINDEX** operations on a table triggers index rebuilding. During this process, data is dumped to a new data file, and once rebuilding is complete, the original file is deleted. For large tables, index rebuilding can consume a significant amount of disk space. Exercise caution when disk space is insufficient to prevent the cluster from becoming read-only.

**WARNING**

- For a table where a large amount of data needs to be added, deleted, or modified, it is recommended that the number of indexes be less than or equal to three. The maximum number of indexes is five.
- Do not perform the **CREATE INDEX** and **REINDEX** operations on large tables during peak hours.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

## Syntax

- Create an index on a table.  

```
CREATE [UNIQUE] INDEX [[IF NOT EXISTS] [schema_name.] index_name] ON table_name
[USING method]
 (({ column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS
{ FIRST | LAST }] }, ...)
 [NULLS [NOT] DISTINCT | NULLS IGNORE]
 [COMMENT 'text']
 [WITH ({ storage_parameter = value } [, ...])]
 [WHERE predicate];
```
- Create an index for a partitioned table.  

```
CREATE [UNIQUE] INDEX [[IF NOT EXISTS] [schema_name.] index_name] ON table_name
[USING method]
 (({ column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS
LAST }] }, ...)
 [NULLS [NOT] DISTINCT | NULLS IGNORE]
 [COMMENT 'text']
 LOCAL [({ PARTITION index_partition_name } [, ...])]
 [WITH ({ storage_parameter = value } [, ...])]
 ;
```

## Parameter Description

**Table 21-204** CREATE INDEX parameters

| Parameter     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range                                                                         |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| UNIQUE        | Causes the system to check for duplicate values in the table when the index is created (if data exists) and each time data is added. If the inserted or updated value causes duplicate records, an error is reported.<br><br>Currently, only B-tree indexes of row-store tables and column-store tables support unique indexes.                                                                                                                   | -                                                                                   |
| IF NOT EXISTS | If <b>IF NOT EXISTS</b> is specified and no index with the same name exists, the index can be created. If an index with the same name already exists, the system will simply display a message stating that the index already exists and will not perform any additional operations during creation. When <b>IF NOT EXISTS</b> is used, the index name must be specified. This parameter is supported only by clusters of version 9.1.0 or later. | -                                                                                   |
| schema_name   | Name of the schema where the index to be created is located. The specified schema name must be the same as the schema of the table.                                                                                                                                                                                                                                                                                                               | -                                                                                   |
| index_name    | Specifies the name of the index to be created. The schema of the index is the same as that of the table.                                                                                                                                                                                                                                                                                                                                          | A string, which must comply with the <a href="#">identifier naming convention</a> . |
| table_name    | Specifies the name of the table to be indexed (optionally schema-qualified).                                                                                                                                                                                                                                                                                                                                                                      | Name of an existing table                                                           |

| Parameter    | Description                                                                                                                                                                                                                                                                                                                             | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| USING method | <p>Specifies the name of the index method to be used.</p> <ul style="list-style-type: none"><li>Row-based tables support the following index types: <b>btree</b> (default), <b>gin</b>, and <b>gist</b>.</li><li>Column-based tables support the following index types: <b>psort</b> (default), <b>btree</b>, and <b>gin</b>.</li></ul> | <p>Value range:</p> <ul style="list-style-type: none"><li><b>btree</b>: B-tree indexes store data keys using a structure like the B+ tree, enabling fast index searches. B-tree indexes support comparison and range queries.</li><li><b>gin</b>: GIN indexes work like reverse lists and handle data with many elements, such as arrays.</li><li><b>gist</b>: GiST indexes work with multidimensional data like geometry, geography, and collections.</li><li><b>psort</b>: PSORT indexes are used to perform partial sort on column-store tables.</li></ul> |
| column_name  | <p>Specifies the name of a column of the table.</p>                                                                                                                                                                                                                                                                                     | <p>Multiple columns can be specified if the index method supports multi-column indexes. A maximum of 32 columns can be specified.</p>                                                                                                                                                                                                                                                                                                                                                                                                                         |
| expression   | <p>Specifies an expression based on one or more columns of the table. The expression usually must be written with surrounding parentheses, as shown in the syntax.</p>                                                                                                                                                                  | <p>However, the parentheses can be omitted if the expression has the form of a function call.</p> <p>Expression can be used to obtain fast access to data based on some transformation of the basic data. For example, an index computed on <code>upper(col)</code> would allow the clause <code>WHERE upper(col) = 'JIM'</code> to use an index.</p> <p>If an expression contains <b>IS NULL</b>, the index for this expression is invalid. In this case, you are advised to create a partial index.</p>                                                     |

| Parameter            | Description                                                                                                                               | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| COLLATE<br>collation | Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used. | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| opclass              | Specifies the name of an operator class.                                                                                                  | Specifies an operator class for each column of an index. The operator class identifies the operators to be used by the index for that column. For example, a B-tree index on the type <code>int4</code> would use the <b><code>int4_ops</code></b> class; this operator class includes comparison functions for values of type <code>int4</code> .<br><br>In practice, the default operator class for the column's data type is sufficient. The operator class applies to data with multiple sorts. For example, we might want to sort a complex-number data type either by absolute value or by real part. We could do this by defining two operator classes for the data type and then selecting the proper class when making an index. |
| ASC                  | Indicates ascending sort order (default). This option is supported only by row storage.                                                   | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| DESC                 | Indicates descending sort order. This option is supported only by row storage.                                                            | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| NULLS FIRST          | Specifies that nulls sort before not-null values. This is the default when <b>DESC</b> is specified.                                      | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| NULLS LAST           | Specifies that nulls sort after not-null values. This is the default when <b>DESC</b> is not specified.                                   | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |



| Parameter                                            | Description                                                                                                                                                                      | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NULLS DISTINCT<br>NULLS NOT DISTINCT<br>NULLS IGNORE | Specifies how NULL values of index columns in a Unique index are processed.                                                                                                      | <p>Default value: This parameter is left empty by default. NULL values can be inserted repeatedly.</p> <p>When the inserted data is compared with the original data in the table, the NULL value can be processed in any of the following ways:</p> <ul style="list-style-type: none"><li>• <b>NULLS DISTINCT:</b> Each NULL is treated as unique, allowing repeated insertions of NULLs.</li><li>• <b>NULLS NOT DISTINCT:</b> All NULLs are seen as identical, so only one NULL is kept in the index, preventing repeated NULL insertions; only distinct non-NULL values can be added.</li><li>• <b>NULLS IGNORE:</b> NULL values are skipped during the equivalent comparison. If all indexed columns are NULL, NULLs can be inserted repeatedly; if some columns are NULL, only new non-NULL values differing from existing ones can be added.</li></ul> <p>The following table <a href="#">Table 21-206</a> lists the behaviors of the three processing modes.</p> |
| COMMENT 'text'                                       | Specifies the comment of an index.                                                                                                                                               | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| WITH<br>( {storage_parameter = value}<br>[, ... ] )  | Specifies the name of an index-method-specific storage parameter. Its value can be <b>FILLFACTOR</b> , <b>INVISIBLE</b> , <b>FASTUPDATE</b> , or <b>GIN_PENDING_LIST_LIMIT</b> . | For details about the value range and usage, see <a href="#">Table 21-205</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

| Parameter                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Value Range                                                                                                                                                                                                             |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WHERE predicate                   | Creates a partial index.<br>A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table. For example, if you have a table that contains both billed and unbilled orders where the unbilled orders take up a small fraction of the total table and yet that is an often used section, you can improve performance by creating an index on just that portion. Another possible application is to use <b>WHERE</b> with <b>UNIQUE</b> to enforce uniqueness over a subset of a table. | Predicate expression can refer only to columns of the underlying table, but it can use all columns, not just the ones being indexed. Presently, subquery and aggregate expressions are also forbidden in <b>WHERE</b> . |
| LOCAL                             | Local indexes (partitioned indexes) are created on each partition of a partitioned table. A local index corresponds to a partition of the partitioned table, and has the same number of partitions and the same partition range as the table. When the table partition changes, you only need to maintain the corresponding index partition.                                                                                                                                                                                                                                          | -                                                                                                                                                                                                                       |
| PARTITION<br>index_partition_name | Specifies the name of the index partition.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | A string, which must comply with the naming convention.                                                                                                                                                                 |

**Table 21-205** WITH ( {storage\_parameter = value} [, ... ] ) parameter

| Storage Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Example                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FILLFACTOR</b> | <p><b>This parameter is unavailable for GIN and PSORT indexes.</b> It indicates the fill factor of an index and its value is a percentage between 10 and 100. Index pages are packed only to the indicated percentage and the remaining space is used for subsequent INSERT and UPDATE operations, thereby reducing page splits and improving performance.</p> <p>This parameter determines the percentage of space on index pages to be packed, which affects the database performance and storage space usage. You can adjust its value to reduce page splits caused by subsequent INSERT and UPDATE operations, optimizing the storage and performance of indexes.</p> <p><b>Impact of FILLFACTOR on index performance:</b></p> <ul style="list-style-type: none"><li>● <b>Performance:</b> When an index page is fully packed and a new record needs to be inserted, the page will be split, consuming more I/O resources and time. A lower fillfactor can reduce page splits and improve the INSERT and UPDATE performance, but will increase the storage space usage. A higher fillfactor can improve the storage space usage, but may cause more page splits and degrade the INSERT and UPDATE performance.</li><li>● <b>Storage space:</b> A lower fillfactor increases the free space of each data page. A higher fillfactor stores data more compactly, reducing disk space usage.</li></ul> | <p>Create a specified B-tree index on the <b>SM_SHIP_MODE_SK</b> column in the <b>tpcds.ship_mode_t1</b> table and set <b>FILLFACTOR</b> of the index to <b>70%</b>.</p> <p>This means that each data page is only 70% full when initially packed. The remaining 30% space is reserved for future INSERT and UPDATE operations.</p> <pre>CREATE INDEX ds_ship_mode_t1_index6 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK) WITH (FILLFACTOR = 70);</pre> |

| Storage Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Example                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                   | <p><b>How to configure FILLFACTOR:</b></p> <ul style="list-style-type: none"><li>For tables that are frequently updated, set <b>FILLFACTOR</b> to a small value, for example, <b>60</b> to <b>80</b>, to reserve space and reduce page splits.</li><li>For tables that are rarely updated, set <b>FILLFACTOR</b> to a large value, for example, <b>90</b> to <b>100</b>, to maximize storage space.</li></ul>                                                                     |                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>INVISIBLE</b>  | <p>Controls whether the optimizer generates index scan plans.</p> <p><b>Scenario:</b> If you want to disable indexes temporarily during maintenance or test the impact of indexes on performance, you can use this parameter.</p> <p><b>Value range:</b></p> <ul style="list-style-type: none"><li><b>ON</b> indicates that no index scan plan is generated.</li><li><b>OFF</b> indicates that an index scan plan is generated.</li></ul> <p><b>Default value:</b> <b>OFF</b></p> | <p>Create a specified B-tree index on the <b>SM_SHIP_MODE_SK</b> column in the <b>tpcds.ship_mode_t1</b> table and set <b>INVISIBLE</b> to <b>ON</b> (<b>ON</b> indicates that optimizer does not generate the index scan plan.)</p> <pre>CREATE INDEX ds_ship_mode_t1_index7 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK) WITH (INVISIBLE = ON);</pre>                                |
| <b>FASTUPDATE</b> | <p><b>This parameter is available only for GIN indexes.</b> It controls whether to enable fast update for GIN indexes. Once enabled, data changes are temporarily stored in the pending list.</p> <p><b>Value range:</b></p> <ul style="list-style-type: none"><li><b>ON:</b> Fast update is enabled.</li><li><b>OFF:</b> Fast update is disabled.</li></ul> <p><b>Default value:</b> <b>ON</b></p>                                                                               | <p>Create a GIN index on the <b>SM_SHIP_MODE_SK</b> column in the <b>tpcds.ship_mode_t1</b> table, enable fast update by configuring the <b>FASTUPDATE</b> parameter, and set the maximum pending list size of the index to 8 MB.</p> <pre>CREATE INDEX ds_ship_mode_t1_index8 ON tpcds.ship_mode_t1 USING gin(SM_SHIP_MODE_SK) WITH (FASTUPDATE = ON, GIN_PENDING_LIST_LIMIT = 8192);</pre> |

| Storage Parameter             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Example |
|-------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <b>GIN_PENDING_LIST_LIMIT</b> | <p><b>This parameter is supported only by GIN indexes.</b> When fast update is enabled for GIN indexes, you can use this parameter to set the maximum size of the pending list of GIN indexes.</p> <p><b>Pending List</b> is a data structure specific to GIN indexes and is used to temporarily store index update operations. When the <b>FASTUPDATE</b> parameter is enabled for a GIN index, new index items are not directly written to the main index structure, but are stored in the pending list. When conditions are met, the index items are merged into the main index in batches.</p> <p><b>Value range:</b> 64 to <i>INT_MAX</i>. The unit is KB.</p> <p><b>Default value:</b> determined by the value of <b>gin_pending_list_limit</b> (4 MB by default).</p> |         |

**Table 21-206** Processing of NULL values in index columns in unique indexes

| Constraint         | All Index Columns Are NULL     | Some Index Columns Are NULL.                                                                               |
|--------------------|--------------------------------|------------------------------------------------------------------------------------------------------------|
| NULLS DISTINCT     | Can be inserted repeatedly.    | Can be inserted repeatedly.                                                                                |
| NULLS NOT DISTINCT | Cannot be inserted repeatedly. | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |
| NULLS IGNORE       | Can be inserted repeatedly.    | Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal. |

## Example

- Create a sample table named **tpcds.ship\_mode\_t1**.

```
CREATE SCHEMA tpcds;
DROP TABLE IF EXISTS tpcds.ship_mode_t1;
CREATE TABLE tpcds.ship_mode_t1
(
 SM_SHIP_MODE_SK INTEGER NOT NULL,
 SM_SHIP_MODE_ID CHAR(16) NOT NULL,
 SM_TYPE CHAR(30) ,
 SM_CODE CHAR(10) ,
 SM_CARRIER CHAR(20) ,
 SM_CONTRACT CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);
```

Create a unique index on the **SM\_SHIP\_MODE\_SK** column in the **tpcds.ship\_mode\_t1** table.

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

Create a unique index on the **SM\_SHIP\_MODE\_SK** column in the **tpcds.ship\_mode\_t1** table and specify how to process null values.

```
CREATE UNIQUE INDEX ds_ship_mode_t1_index5 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK)
NULLS NOT DISTINCT;
```

Add comment to the index when creating an index on the **SM\_SHIP\_MODE\_SK** column of table **tpcds.ship\_mode\_t1**.

```
CREATE INDEX ds_ship_mode_t1_index_comment ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK)
COMMENT 'index';
```

Create a B-tree index on the **SM\_SHIP\_MODE\_SK** column in the **tpcds.ship\_mode\_t1** table.

```
CREATE INDEX ds_ship_mode_t1_index4 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK);
```

Create a specified B-tree index on the **SM\_SHIP\_MODE\_SK** column in the **tpcds.ship\_mode\_t1** table and set **FILLFACTOR** of the index to **70%**.

```
CREATE INDEX ds_ship_mode_t1_index6 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK)
WITH (FILLFACTOR = 70);
```

Create a specified B-tree index on the **SM\_SHIP\_MODE\_SK** column in the **tpcds.ship\_mode\_t1** table and set **INVISIBLE** to **ON** (**ON** indicates that optimizer does not generate the index scan plan. The default value is **OFF**).

```
CREATE INDEX ds_ship_mode_t1_index7 ON tpcds.ship_mode_t1 USING btree(SM_SHIP_MODE_SK)
WITH (INVISIBLE = ON);
```

Create a GIN index on the **SM\_SHIP\_MODE\_SK** column in the **tpcds.ship\_mode\_t1** table, enable fast update by configuring the **FASTUPDATE** parameter, and set the maximum pending list size of the index to 8 MB.

```
CREATE INDEX ds_ship_mode_t1_index8 ON tpcds.ship_mode_t1 USING gin(SM_SHIP_MODE_SK)
WITH (FASTUPDATE = ON, GIN_PENDING_LIST_LIMIT = 8192);
```

Create an expression index on the **SM\_CODE** column in the **tpcds.ship\_mode\_t1** table.

```
CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1,4));
```

Create a partial index on the **SM\_SHIP\_MODE\_SK** column where **SM\_SHIP\_MODE\_SK** is greater than **10** in the **tpcds.ship\_mode\_t1** table.

```
CREATE UNIQUE INDEX ds_ship_mode_t1_index3 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK)
WHERE SM_SHIP_MODE_SK>10;
```

- Create a sample table named **tpcds.customer\_address\_p1**.

```
CREATE TABLE tpcds.customer_address_p1
(
 CA_ADDRESS_SK INTEGER NOT NULL,
 CA_ADDRESS_ID CHAR(16) NOT NULL,
```

```
CA_STREET_NUMBER CHAR(10) ,
CA_STREET_NAME VARCHAR(60) ,
CA_STREET_TYPE CHAR(15) ,
CA_SUITE_NUMBER CHAR(10) ,
CA_CITY VARCHAR(60) ,
CA_COUNTY VARCHAR(30) ,
CA_STATE CHAR(2) ,
CA_ZIP CHAR(10) ,
CA_COUNTRY VARCHAR(20) ,
CA_GMT_OFFSET DECIMAL(5,2) ,
CA_LOCATION_TYPE CHAR(20)
)
DISTRIBUTE BY HASH(CA_ADDRESS_SK)
PARTITION BY RANGE(CA_ADDRESS_SK)
(
 PARTITION p1 VALUES LESS THAN (3000),
 PARTITION p2 VALUES LESS THAN (5000) ,
 PARTITION p3 VALUES LESS THAN (MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

Create the partitioned table index **ds\_customer\_address\_p1\_index1** with the name of the index partition not specified.

```
CREATE INDEX ds_customer_address_p1_index1 ON tpcds.customer_address_p1(CA_ADDRESS_SK)
LOCAL;
```

Create the partitioned table index **ds\_customer\_address\_p1\_index2** with the name of the index partition specified.

```
CREATE INDEX ds_customer_address_p1_index2 ON tpcds.customer_address_p1(CA_ADDRESS_SK)
LOCAL
(
 PARTITION CA_ADDRESS_SK_index1,
 PARTITION CA_ADDRESS_SK_index2,
 PARTITION CA_ADDRESS_SK_index3
)
;
```

Create the partitioned table index **ds\_customer\_address\_p1\_index\_comment** and add index comments.

```
CREATE INDEX ds_customer_address_p1_index_comment ON
tpcds.customer_address_p1(CA_ADDRESS_SK) COMMENT 'index' LOCAL
(
 PARTITION CA_ADDRESS_SK_index1,
 PARTITION CA_ADDRESS_SK_index2,
 PARTITION CA_ADDRESS_SK_index3
)
;
```

## Application Differences Among NULLS DISTINCT, NULLS NOT DISTINCT, and NULLS IGNORE

Examples are provided below to illustrate the differences.

### Example

Assume that there is a **users** table that contains ID and Email (which may be NULL, indicating that no email address is provided).

#### Scenario 1: NULLS DISTINCT (NULL can be inserted repeatedly) (default)

```
--Create a table.
DROP TABLE IF EXISTS users;
CREATE TABLE users (
 id INT,
 email TEXT
```

```
)
DISTRIBUTE BY HASH(email);
--Create a unique index (NULLS DISTINCT by default).
CREATE UNIQUE INDEX idx_email_unique ON users (email);
--Insert data.
INSERT INTO users (id, email) VALUES (1, NULL); -- Success
INSERT INTO users (id, email) VALUES (2, NULL); -- Success (with each NULL treated as unique)
INSERT INTO users (id, email) VALUES (3, 'a@example.com'); -- Success
INSERT INTO users (id, email) VALUES (4, 'a@example.com'); -- Failure (duplicate non-NULL values)
```

```
error_code: DWS.S0010010
error_msg: Constraint Violation. STATE: 23505, message: ERROR: dn_6005_6006: duplicate key value
violates unique constraint "idx_email_unique"
Detail: Key (email)=(a@example.com) already exists.
```

### Scenario 2: NULLS NOT DISTINCT (NULL cannot be inserted repeatedly)

```
--Re-create a unique index and specify NULLS NOT DISTINCT.
DROP TABLE IF EXISTS users;
CREATE TABLE users (
 id INT,
 email TEXT
)
DISTRIBUTE BY HASH(email);
CREATE UNIQUE INDEX idx_email_unique ON users (email) NULLS NOT DISTINCT;
--Insert data.
INSERT INTO users (id, email) VALUES (5, NULL); -- Success (the first NULL)
INSERT INTO users (id, email) VALUES (6, NULL); -- Failure (the second NULL, violating the unique
constraint)
INSERT INTO users (id, email) VALUES (7, 'b@example.com'); -- Success
INSERT INTO users (id, email) VALUES (8, 'b@example.com'); -- Failure (duplicate non-NULL values)
```

### Scenario 3: NULLS IGNORE (NULL values are ignored.)

```
--Create an index and ignore NULL values.
DROP TABLE IF EXISTS users;
CREATE TABLE users (
 id INT,
 email TEXT
)
DISTRIBUTE BY HASH(email);
CREATE UNIQUE INDEX idx_email_ignore ON users (email) NULLS IGNORE;
--Insert data.
INSERT INTO users (id, email) VALUES (9, NULL); -- Success (the first NULL)
INSERT INTO users (id, email) VALUES (10, NULL); -- Success (NULL values are ignored.)
INSERT INTO users (id, email) VALUES (11, 'c@example.com'); -- Success
INSERT INTO users (id, email) VALUES (12, 'c@example.com'); -- Failure (duplicate non-NULL values)
```

## Helpful Links

- For details about how to modify and delete indexes, see [ALTER INDEX](#) and [DROP INDEX](#).
- For more index examples, see "Creating and Managing Indexes" in the *Data Warehouse Service (DWS) Developer Guide*.

## 21.13.6.2 ALTER INDEX

### Function

**ALTER INDEX** modifies the definition of an index.



## Precautions

Only the owner of an index or a system administrator can run this statement.

## Syntax

- Rename a table index. The new index name can be prefixed with the name of the schema where the original index is located, but the schema name cannot be changed at the same time.

```
ALTER INDEX [IF EXISTS] index_name
 RENAME TO new_name;
ALTER INDEX [IF EXISTS] index_name
 RENAME TO schema.new_name;
```

- Modify the storage parameter of a table index.

```
ALTER INDEX [IF EXISTS] index_name
 SET ({storage_parameter = value} [, ...]);
```

- Modify the status flag of an index.

```
ALTER INDEX [IF EXISTS] index_name
 SET ({invisible = value} [, ...]);
```

- Reset the storage parameter of a table index.

```
ALTER INDEX [IF EXISTS] index_name
 RESET (storage_parameter [, ...]);
```

- Set a table index or an index partition to be unavailable. **The syntax cannot be used for column-store tables.**

```
ALTER INDEX [IF EXISTS] index_name
 [MODIFY PARTITION index_partition_name] UNUSABLE;
```

- Rebuild a table index or index partition.

```
ALTER INDEX index_name
 REBUILD [PARTITION index_partition_name] [WITHOUT UNUSABLE];
```

- Rename an index partition. When **PG\_OBJECT** record the last modification time of an index, this syntax cannot be recorded.

```
ALTER INDEX [IF EXISTS] index_name
 RENAME PARTITION index_partition_name TO new_index_partition_name;
```

- Add and modify the index comment.

```
ALTER INDEX [IF EXISTS] index_name
 COMMENT 'text';
```

- Delete the index comment.

```
ALTER INDEX [IF EXISTS] index_name
 COMMENT '';
ALTER INDEX [IF EXISTS] index_name
 COMMENT NULL;
```

## Parameter Description

Table 21-207 ALTER INDEX parameters

| Parameter | Description                                                                      | Value Range |
|-----------|----------------------------------------------------------------------------------|-------------|
| IF EXISTS | If the specified index does not exist, a prompt instead of an error is returned. | -           |
| RENAME TO | Renames an index. It has no impact on the stored data.                           | -           |

| Parameter                                                | Description                                                                                                                                                                                                                            | Value Range                                                                     |
|----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| SET<br>( { STORAGE_PARAMETER<br>= value } [, ...] )      | Changes one or more index-method-specific storage parameters. This command does not change the index content right away. Depending on the settings, you might need to rebuild the index with <b>REINDEX</b> to get the desired result. | For details about the value range and usage, see <a href="#">Table 21-208</a> . |
| RESET<br>( { storage_parameter }<br>[, ...] )            | Resets one or more index method-specific storage parameters of an index to default values. Similar to the <b>SET</b> statement, <b>REINDEX</b> may be used to completely update the index.                                             | -                                                                               |
| [ MODIFY PARTITION<br>index_partition_name ]<br>UNUSABLE | Sets the index on a table or index partition to be unavailable.                                                                                                                                                                        | -                                                                               |
| REBUILD [ PARTITION<br>index_partition_name ]            | Recreates the index on a table or index partition.                                                                                                                                                                                     | -                                                                               |
| WITHOUT UNUSABLE                                         | Indexes that are in an UNUSABLE state are not rebuilt when rebuilding indexes in tables or index partitions.                                                                                                                           | -                                                                               |
| RENAME PARTITION                                         | Renames an index partition.                                                                                                                                                                                                            | -                                                                               |
| COMMENT comment_text                                     | Adds, modifies, or deletes index comments.                                                                                                                                                                                             | -                                                                               |
| index_name                                               | Specifies the index name to be modified.                                                                                                                                                                                               | Name of an existing index.                                                      |
| new_name                                                 | Specifies the new name for the index.                                                                                                                                                                                                  | A string compliant with the <a href="#">identifier naming rules</a> .           |
| storage_parameter                                        | Specifies the name of an index-method-specific parameter.                                                                                                                                                                              | -                                                                               |

| Parameter                | Description                                                                                                                          | Value Range                                                                                                                                                                                                |
|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| invisible                | Controls whether the optimizer generates index scan plans. The default value is <b>OFF</b> .                                         | <b>ON</b> or <b>OFF</b> . <ul style="list-style-type: none"><li>• <b>ON</b> indicates that no index scan plan is generated.</li><li>• <b>OFF</b> indicates that an index scan plan is generated.</li></ul> |
| value                    | Specifies the new value for an index-method-specific storage parameter. This might be a number or a word depending on the parameter. | -                                                                                                                                                                                                          |
| new_index_partition_name | Specifies the new name of the index partition.                                                                                       | A string compliant with the <a href="#">identifier naming rules</a> .                                                                                                                                      |
| index_partition_name     | Specifies the name of the index partition.                                                                                           | A string compliant with the <a href="#">identifier naming rules</a> .                                                                                                                                      |
| comment_text             | Specifies the comment of an index.                                                                                                   | -                                                                                                                                                                                                          |

**Table 21-208** WITH ( {storage\_parameter = value} [, ... ] ) parameter description

| Storage Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Example                                                                                                                             |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| FILLFACTOR        | <p><b>This parameter is unavailable for GIN and PSORT indexes.</b> It indicates the fill factor of an index and its value is a percentage between 10 to 100. Index pages are packed only to the indicated percentage and the remaining space is used for subsequent INSERT and UPDATE operations, thereby reducing page splits and improving performance.</p> <p>This parameter determines the percentage of space on index pages to be packed, which affects the database performance and storage space usage. You can adjust its value to reduce page splits caused by subsequent INSERT and UPDATE operations, optimizing the storage and performance of indexes.</p> <p><b>Impact of FILLFACTOR on index performance:</b></p> <ul style="list-style-type: none"><li>● <b>Performance:</b> When an index page is fully packed and a new record needs to be inserted, the page will be split, consuming more I/O resources and time. A lower fillfactor can reduce page splits and improve the INSERT and UPDATE performance, but will increase the storage space usage. A higher fillfactor can improve the storage space usage, but may cause more page splits and degrade the INSERT and UPDATE performance.</li><li>● <b>Storage space:</b> A lower fillfactor increases the free space of each data page. A higher fillfactor stores data more compactly, reducing disk space usage.</li></ul> | <p>Change the fillfactor of an index to 60%.</p> <pre>ALTER INDEX<br/>tpcds.ds_ship_mode_t1_index7<br/>SET (FILLFACTOR = 60);</pre> |

| Storage Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Example                                                                                                                                                                                                                 |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                   | <p><b>How to configure FILLFACTOR:</b></p> <ul style="list-style-type: none"><li>For tables that are frequently updated, set <b>FILLFACTOR</b> to a small value, for example, <b>60</b> to <b>80</b>, to reserve space and reduce page splits.</li><li>For tables that are rarely updated, set <b>FILLFACTOR</b> to a large value, for example, <b>90</b> to <b>100</b>, to maximize storage space.</li></ul>                                                              |                                                                                                                                                                                                                         |
| <b>INVISIBLE</b>  | <p>Controls whether the optimizer generates index scan plans.</p> <p><b>Scenario:</b> If you want to disable indexes temporarily during maintenance or test the impact of indexes on performance, you can use this parameter.</p> <p><b>Value range:</b></p> <ul style="list-style-type: none"><li><b>ON</b> indicates that no index scan plan is generated.</li><li><b>OFF</b> indicates that an index scan plan is generated.</li></ul> <p><b>Default value: OFF</b></p> | <p>Not allow the optimizer to generate any index scan plan.</p> <pre>ALTER INDEX<br/>tpcds.ds_ship_mode_t1_index9<br/>SET (INVISIBLE = ON);</pre>                                                                       |
| <b>FASTUPDATE</b> | <p><b>This parameter is available only for GIN indexes.</b> It controls whether to enable fast update for GIN indexes. Once enabled, data changes are temporarily stored in the pending list.</p> <p><b>Value range: ON or OFF</b></p> <p><b>Default value: ON</b></p>                                                                                                                                                                                                     | <p>Enable fast update and set the maximum size of the pending list of the index to 8 MB.</p> <pre>ALTER INDEX<br/>tpcds.ds_ship_mode_t1_index10<br/>SET (FASTUPDATE = ON,<br/>GIN_PENDING_LIST_LIMIT =<br/>8192);</pre> |

| Storage Parameter             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Example |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <b>GIN_PENDING_LIST_LIMIT</b> | <p><b>This parameter is supported only by GIN indexes.</b> When fast update is enabled for GIN indexes, you can use this parameter to set the maximum size of the pending list of GIN indexes.</p> <p>The pending list is a data structure specific to GIN indexes and is used to temporarily store index update operations. When <b>FASTUPDATE</b> is enabled for a GIN index, new index items are not directly written to the main index structure, but are stored in the pending list. When conditions are met, the index items are merged into the main index in batches.</p> <p><b>Value range:</b> 64 to <i>INT_MAX</i>.<br/>The unit is KB.</p> <p><b>Default value:</b> determined by the value of <b>gin_pending_list_limit</b> (4 MB by default).</p> |         |

## Examples

Create a sample table named **tpcds.ship\_mode\_t1**.

```
DROP TABLE IF EXISTS tpcds.ship_mode_t1;
CREATE TABLE tpcds.ship_mode_t1
(
 SM_SHIP_MODE_SK INTEGER NOT NULL,
 SM_SHIP_MODE_ID CHAR(16) NOT NULL,
 SM_TYPE CHAR(30) ,
 SM_CODE CHAR(10) ,
 SM_CARRIER CHAR(20) ,
 SM_CONTRACT CHAR(20)
)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);
```

Create a unique index on the **SM\_SHIP\_MODE\_SK** column in the **tpcds.ship\_mode\_t1** table.

```
DROP INDEX IF EXISTS ds_ship_mode_t1_index1;
CREATE UNIQUE INDEX ds_ship_mode_t1_index1 ON tpcds.ship_mode_t1(SM_SHIP_MODE_SK);
```

Create an expression index on the **SM\_CODE** column in the **tpcds.ship\_mode\_t1** table.

```
DROP INDEX IF EXISTS ds_ship_mode_t1_index2;
CREATE INDEX ds_ship_mode_t1_index2 ON tpcds.ship_mode_t1(SUBSTR(SM_CODE,1,4));
```

Rename the **ds\_ship\_mode\_t1\_index1** index to **ds\_ship\_mode\_t1\_index5**.

```
ALTER INDEX tpcds.ds_ship_mode_t1_index1 RENAME TO ds_ship_mode_t1_index5;
```

Set the **ds\_ship\_mode\_t1\_index2** index as unusable.

```
ALTER INDEX tpcds.ds_ship_mode_t1_index2 UNUSABLE;
```

Rebuild the **ds\_ship\_mode\_t1\_index2** index.

```
ALTER INDEX tpcds.ds_ship_mode_t1_index2 REBUILD;
```

## Links

[CREATE INDEX](#), [DROP INDEX](#), and [REINDEX](#)

### 21.13.6.3 REINDEX

#### Function

**REINDEX** rebuilds an index using the data stored in the index's table, replacing the old copy of the index.

There are several scenarios in which **REINDEX** can be used:

- An index has become corrupted, and no longer contains valid data.
- An index has become "bloated", that is, it contains many empty or nearly-empty pages.
- You have altered a storage parameter (such as fillfactor) for an index, and wish to ensure that the change has taken full effect.

An index build with the **CONCURRENTLY** option failed, leaving an "invalid" index.

#### Precautions

Index reconstruction of the **REINDEX DATABASE** or **SYSTEM** type cannot be performed in transaction blocks.

In the storage-compute decoupling architecture, REINDEX full database indexes are not supported.



#### WARNING

- Do not perform the **CREATE INDEX** and **REINDEX** operations on large tables during peak hours.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

#### Syntax

- Recreate indexes for the entire database or system catalogs.  

```
REINDEX { DATABASE | SYSTEM } name [FORCE];
```

- Recreate an ordinary index or an index on an ordinary table.  
`REINDEX { INDEX | TABLE } name [ FORCE | WITHOUT UNUSABLE ]`
- Recreate an index partition or an index on a partition.  
`REINDEX { | TABLE } name  
PARTITION partition_name [ FORCE | WITHOUT UNUSABLE ];`

## Parameter Description

Table 21-209 REINDEX parameters

| Parameter        | Description                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                        |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| INDEX            | Recreates the specified index.                                                                                                                                                                                                                                                                     | -                                                                                                                                                                                                  |
| TABLE            | Recreates all indexes of the specified table. If the table has a secondary TOAST table, that is reindexed as well.                                                                                                                                                                                 | -                                                                                                                                                                                                  |
| DATABASE         | Recreates all indexes within the current database.                                                                                                                                                                                                                                                 | -                                                                                                                                                                                                  |
| SYSTEM           | Recreates all indexes on system catalogs within the current database. Indexes on user tables are not processed.                                                                                                                                                                                    | -                                                                                                                                                                                                  |
| name             | Name of the specific index, table, or database to be reindexed. Index and table names can be schema-qualified.<br><b>NOTE</b><br><b>REINDEX DATABASE</b> and <b>SYSTEM</b> can create indexes for only the current database. Therefore, <b>name</b> must be the same as the current database name. | -                                                                                                                                                                                                  |
| FORCE            | This is an obsolete option. It is ignored if specified.                                                                                                                                                                                                                                            | -                                                                                                                                                                                                  |
| WITHOUT UNUSABLE | If this parameter is specified, the <b>UNUSABLE</b> indexes or <b>UNUSABLE</b> index partitions on the object is ignored during <b>REINDEX</b> . This parameter is supported only by 9.1.0.100 and later versions.                                                                                 | -                                                                                                                                                                                                  |
| partition_name   | Specifies the name of the partition or index partition to be reindexed.                                                                                                                                                                                                                            | <ul style="list-style-type: none"><li>• If it is <b>REINDEX INDEX</b>, specify the name of an index partition.</li><li>• If it is <b>REINDEX TABLE</b>, specify the name of a partition.</li></ul> |



**NOTICE**

Index reconstruction of the **REINDEX DATABASE** or **SYSTEM** type cannot be performed in transaction blocks.

## Examples

Rebuild a single index.

```
REINDEX INDEX tpcds.tpcds_customer_index1;
```

Rebuild all indexes on the **tpcds.customer\_t1** table.

```
REINDEX TABLE tpcds.customer_t1;
```

Rebuild the partition index of partition **P1** in the partitioned table **customer\_address**.

```
DROP TABLE IF EXISTS customer_address;
CREATE TABLE 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)
)
DISTRIBUTE BY HASH (ca_address_sk)
PARTITION BY RANGE(ca_address_sk)
(
 PARTITION P1 VALUES LESS THAN(2450815),
 PARTITION P2 VALUES LESS THAN(2451179),
 PARTITION P3 VALUES LESS THAN(2451544),
 PARTITION P4 VALUES LESS THAN(MAXVALUE)
);

CREATE INDEX customer_address_index on customer_address(CA_ADDRESS_SK) LOCAL;
REINDEX TABLE customer_address PARTITION P1;
```

### 21.13.6.4 DROP INDEX

#### Function

**DROP INDEX** deletes an index.

#### Precautions

Only the owner of an index or a system administrator can run **DROP INDEX** command.

#### Syntax

```
DROP INDEX [CONCURRENTLY][IF EXISTS]
index_name [, ...] [CASCADE | RESTRICT];
```

## Parameter Description

**Table 21-210** DROP INDEX parameters

| Parameter          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                                                               |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONCURRENTLY       | Deletes an index without locking concurrent selections, inserts, updates, and deletes on the index table. A normal <b>DROP INDEX</b> obtains an exclusive lock on the table to prevent other access until the index is deleted. With this option, the command waits until the conflicting transaction is complete.<br><b>NOTE</b><br>Note that only one index name can be specified and the <b>CASCADE</b> option is not supported. (Therefore, indexes that support <b>UNIQUE</b> or <b>PRIMARY KEY</b> constraints cannot be deleted in this way.) Regular <b>DROP INDEX</b> commands can be executed within a transaction block, but cannot be executed in <b>DROP INDEX CONCURRENTLY</b> mode. | -                                                                                                                                                                                                                                         |
| IF EXISTS          | Sends a notice instead of an error if the specified index does not exist.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | -                                                                                                                                                                                                                                         |
| index_name         | Specifies the name of the index to be deleted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | An existing index.                                                                                                                                                                                                                        |
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: automatically deletes all objects that depend on the index to be deleted.</li><li>• <b>RESTRICT</b> (default): refuses to delete the index if any objects depend on it.</li></ul> |

## Examples

Delete the **ds\_ship\_mode\_t1\_index2** index.

```
DROP INDEX tpcds.ds_ship_mode_t1_index2;
```

## Helpful Links

[ALTER INDEX](#), [CREATE INDEX](#)

## 21.13.6.5 CREATE SEQUENCE

### Function

**CREATE SEQUENCE** adds a sequence to the current database.

A sequence is a database object that generates a series of unique, ascending, or descending values. It is mainly used to generate the primary key or unique identifier in a table.

### Precautions

- The user who creates a sequence is the owner of the sequence.
- If a schema name is specified, the sequence is created in the specified schema. Otherwise, the text search configuration is created in the current schema. The sequence name must be different from the names of other sequences, tables, indexes, views in the same schema.
- After the sequence is created, functions `nextval()` and `generate_series(1,N)` insert data to the table. Make sure that the number of times for invoking `nextval` is greater than or equal to `N+1`. Otherwise, errors will be reported because the number of times for invoking function `generate_series()` is `N+1`.
- A sequence cannot be created in the **template1** database.
- You are advised to control the number of sequences (including the SERIAL type). In distributed scenarios, sequence values are generated based on GTM. Frequent interaction with GTM severely affects performance. You are advised to reference the public sequence object or use UUIDs for unique values. To minimize the interaction with GTM, you are advised to set **CACHE** of a manually created sequence to a value greater than or equal to 10000.

### Syntax

```
CREATE SEQUENCE [IF NOT EXISTS] name [INCREMENT [BY] increment]
 [MINVALUE minvalue | NO MINVALUE | NOMINVALUE] [MAXVALUE maxvalue | NO MAXVALUE |
NOMAXVALUE]
 [START [WITH] start] [CACHE cache] [[NO] CYCLE | NOCYCLE]
 [OWNED BY { table_name.column_name | NONE }];
```

## Parameter Description

**Table 21-211** CREATE SEQUENCE parameters

| Parameter     | Description                                                                                                                                                                                                                                                                                                                                                                                                                           | Value Range                                                           |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| IF NOT EXISTS | If <b>IF NOT EXISTS</b> is specified and a sequence with the same name does not exist, the sequence can be created successfully. If a sequence with the same name already exists during sequence creation, the system will display a message indicating that the sequence already exists and no further operations will be performed. No error will be reported.<br><br>This parameter is supported only by 9.1.0 and later versions. | -                                                                     |
| name          | Specifies the name of the sequence to be created.                                                                                                                                                                                                                                                                                                                                                                                     | A string compliant with the <a href="#">identifier naming rules</a> . |
| increment     | Specifies the step for a sequence. If the increment is a positive value, an increasing sequence is to be created. If the increment is a negative value, a decreasing sequence is to be created.<br><br>The default value is <b>1</b> .                                                                                                                                                                                                | -                                                                     |

| Parameter                                            | Description                                                                                                                                                                                                                                                                                                                            | Value Range |
|------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| MINVALUE<br>minvalue   NO<br>MINVALUE <br>NOMINVALUE | Specifies the minimum value that can be generated by the sequence. If <b>MINVALUE</b> is not specified, or <b>NO MINVALUE</b> is specified, the default value of the ascending sequence is <b>1</b> , and that of the descending sequence is <b>-2<sup>63</sup>-1</b> .<br><br><b>NOMINVALUE</b> is equivalent to <b>NO MINVALUE</b> . | -           |
| MAXVALUE<br>maxvalue   NO<br>MAXVALUE <br>NOMAXVALUE | Specifies the maximum value that can be generated by the sequence. If <b>MAXVALUE</b> is not declared or <b>NO MAXVALUE</b> is declared, the default value of the ascending sequence is <b>2<sup>63</sup>-1</b> , and that of the descending sequence is <b>-1</b> .<br><br><b>NOMAXVALUE</b> is equivalent to <b>NO MAXVALUE</b> .    | -           |
| start                                                | Specifies the start value of the sequence.<br><br>The default value for ascending sequences is <b>minvalue</b> and for descending sequences <b>maxvalue</b> .                                                                                                                                                                          | -           |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| cache     | <p>Specifies the number of sequence numbers that are cached (pre-allocated) to the memory in a database for faster access. Within a cache period, the CN does not request a sequence number from the GTM. Instead, the CN uses the sequence number that is locally applied for in advance.</p> <p>In cluster versions 9.1.0.100 and later, the default value is specified by the GUC parameter <b>default_sequence_cache</b>.</p> <ul style="list-style-type: none"><li>• In a newly installed cluster of 9.1.0.100 or later, the default value is <b>20</b>.</li><li>• If the cluster is upgraded from an earlier version to 9.1.0.100 or later, the default value is <b>1</b>. Indicates that only one value can be generated at a time, that is, no cache.</li></ul> | <ul style="list-style-type: none"><li>• It is not recommended that you define <b>cache</b>, and <b>maxvalue</b>, and <b>minvalue</b> at the same time. The continuity of sequences cannot be ensured after <b>cache</b> is defined because unacknowledged sequences may be generated, wasting sequence number segments.</li><li>• You are advised not to set a large value for <b>cache</b> (less than 100000000). Otherwise, it takes a long time to cache the sequence number (the first <b>NEXTVAL</b> in each cache period). Set a proper value for <b>cache</b> based on services to ensure quick access without wasting sequence numbers.</li></ul> |

| Parameter       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CYCLE   NOCYCLE | <p>Recycles sequences after the number of sequences reaches <b>maxvalue</b> or <b>minvalue</b>.</p> <p>The default value is <b>NO CYCLE</b>, indicating that no new value can be generated after the sequence reaches the maximum or minimum value. <b>NOCYCLE</b> is equivalent to <b>NO CYCLE</b>.</p> <ul style="list-style-type: none"><li>• If the sequence is defined as <b>CYCLE</b>, the sequence uniqueness cannot be ensured.</li><li>• If <b>NO CYCLE</b> is specified, any invocation of <b>nextval</b> would return an error after the sequence reaches its maximum value.</li></ul> | -                                                                                                                                                                                                                                                                                                                                                                                        |
| OWNED BY        | <p>Associates a sequence with a specified column of a specific table. If the column or the table where the column is located is deleted, the sequence is automatically deleted. The associated table and sequence must be owned by the same user and in the same schema.</p> <p>If the default value is <b>OWNED BY NONE</b>, indicating that such association does not exist.</p>                                                                                                                                                                                                                | <ul style="list-style-type: none"><li>• <b>OWNED BY</b> only establishes the association between a table column and the sequence. The sequence is not created for this column.</li><li>• You are not advised to use the sequence created using <b>OWNED BY</b> in other tables. If multiple tables need to share a sequence, the sequence must not belong to a specific table.</li></ul> |

## Examples

Create an ascending sequence named **serial**, which starts from 101:

```
CREATE SEQUENCE serial
START 101
CACHE 20;
```

Select the next number from the sequence:

```
SELECT nextval('serial');
nextval

101
```

Select the next number from the sequence:

```
SELECT nextval('serial');
nextval

102
```

Create a sequence associated with the table:

```
CREATE TABLE customer_address
(
 ca_address_sk integer not null,
 ca_address_id char(16) not null,
 ca_street_number char(10) ,
 ca_street_name varchar(60) ,
 ca_street_type char(15) ,
 ca_suite_number char(10) ,
 ca_city varchar(60) ,
 ca_county varchar(30) ,
 ca_state char(2) ,
 ca_zip char(10) ,
 ca_country varchar(20) ,
 ca_gmt_offset decimal(5,2) ,
 ca_location_type char(20)
);

CREATE SEQUENCE serial1
START 101
CACHE 20
OWNED BY customer_address.ca_address_sk;
```

Use **SERIAL** to create a serial table **serial\_table** for primary key auto-increment.

```
CREATE TABLE serial_table(a int, b serial);
INSERT INTO serial_table (a) VALUES (1),(2),(3);
SELECT * FROM serial_table ORDER BY b;
a | b
---+---
1 | 1
2 | 2
3 | 3
(3 rows)
```

## Helpful Links

[ALTER SEQUENCE, DROP SEQUENCE](#)

### 21.13.6.6 ALTER SEQUENCE

#### Function

Modifies the sequence definition.

#### Precautions

- You must be the owner of the sequence to use **ALTER SEQUENCE**.



- To change the owner, you must also be a direct or indirect member of the new role, and this role must have the **CREATE** permission on the sequence's schema.
- In the current version, you can modify only the owner, home column, and the maximum value. To modify other parameters, delete the sequence and create it again. Then, use the Setval function to restore original parameter values.
- **ALTER SEQUENCE MAXVALUE** cannot be used in transactions, functions, and stored procedures.
- After the maximum value of a sequence is changed, the cache of the sequence in all sessions is cleared.
- **ALTER SEQUENCE** blocks the invocation of **nextval**, **setval**, **curval**, and **lastval**.

## Syntax

Change the maximum value or home column of the sequence.

```
ALTER SEQUENCE [IF EXISTS] name
 [MAXVALUE maxvalue | NO MAXVALUE | NOMAXVALUE]
 [CACHE cache]
 [OWNED BY { table_name.column_name | NONE }] ;
```

Change the owner of the sequence.

```
ALTER SEQUENCE [IF EXISTS] name OWNER TO new_owner;
```

## Parameter Description

Table 21-212 ALTER SEQUENCE parameters

| Parameter                             | Description                                                                                                                                                                                                                                      | Value Range                   |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|
| IF EXISTS                             | Sends a message instead of an error if the sequence does not exist.                                                                                                                                                                              | -                             |
| name                                  | Specifies the name of the sequence to be modified.                                                                                                                                                                                               | Name of an existing sequence. |
| MAXVALUE<br>maxvalue   NO<br>MAXVALUE | Specifies the maximum value of a sequence. If <b>NO MAXVALUE</b> is specified, the default value of the ascending sequence is $2^{63}-1$ , and that of the descending sequence is $-1$ . <b>NOMAXVALUE</b> is equivalent to <b>NO MAXVALUE</b> . | -                             |

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Value Range                                                                                                                                                                                                                                      |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CACHE cache | Allocates sequence numbers in advance and stores them in memory to speed up access. This parameter is supported only by clusters of version 8.2.1.100 or later.                                                                                                                                                                                                                                                                                                                             | <ul style="list-style-type: none"><li>The minimum value of <b>cachevalue</b> is <b>1</b>. (One value can be generated at a time, that is <b>NOCACHE</b>.)</li><li>If this parameter is not specified, the old cache value is retained.</li></ul> |
| OWNED BY    | Associates a sequence with a specified field in a table. In this way, the sequence will be deleted when you delete its associated field or the table where the field belongs.<br><br>If the sequence has been associated with another table before you use this parameter, the new association will overwrite the old one. The associated table and sequence must be owned by the same user and in the same schema. If <b>OWNED BY NONE</b> is used, existing associations will be deleted. | -                                                                                                                                                                                                                                                |
| new_owner   | Specifies the username of the new owner.                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Name of an existing user.                                                                                                                                                                                                                        |

## Examples

Create the sample sequence **seq\_test** and sample table **t1**.

```
CREATE SEQUENCE seq_test START 101;
DROP TABLE IF EXISTS t1;
CREATE TABLE t1(c1 bigint default nextval('seq_test'));
```

Modify the maximum value of **serial** to **200**.

```
ALTER SEQUENCE seq_test MAXVALUE 200;
```

Change the home column of **seq\_test** to **t1.c1**.

```
ALTER SEQUENCE seq_test OWNED BY t1.c1;
```

## Helpful Links

[CREATE SEQUENCE](#) and [DROP SEQUENCE](#)

## 21.13.6.7 DROP SEQUENCE

### Function

**DROP SEQUENCE** deletes a sequence from the current database.

### Precautions

Only a sequence owner or a system administrator can delete a sequence.

### Syntax

```
DROP SEQUENCE [IF EXISTS] {[schema.]sequence_name} [, ...] [CASCADE | RESTRICT];
```

### Parameter Description

**Table 21-213** DROP SEQUENCE parameters

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF EXISTS          | Sends a notice instead of an error if the specified sequence does not exist.                        | -                                                                                                                                                                                                                                               |
| name               | Specifies the name of the sequence to be deleted.                                                   | Specifies an existing sequence name.                                                                                                                                                                                                            |
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• CASCADE: Automatically deletes objects that depend on the sequence to be deleted.</li><li>• RESTRICT: Refuses to delete the sequence if any objects depend on it. This is the default action.</li></ul> |

### Examples

Delete sequence **serial**.

```
DROP SEQUENCE serial;
```

### Helpful Links

[ALTER SEQUENCE](#), [CREATE SEQUENCE](#)

## 21.13.6.8 CREATE TEXT SEARCH CONFIGURATION

### Function

**CREATE TEXT SEARCH CONFIGURATION** creates a text search configuration. A text search configuration specifies a text search parser that can divide a string into

tokens, plus dictionaries that can be used to determine which tokens are of interest for searching.

## Precautions

- If only the parser is specified, then the new text search configuration initially has no mappings from token types to dictionaries, and therefore will ignore all words. Subsequent **ALTER TEXT SEARCH CONFIGURATION** commands must be used to create mappings to make the configuration useful. If **COPY** option is specified, the parser, mapping and configuration option of text search configuration is copied automatically.
- If a schema name is specified, the text search configuration is created in the specified schema. Otherwise, the text search configuration is created in the current schema.
- The user who defines a text search configuration becomes its owner.
- **PARSER** and **COPY** options are mutually exclusive, because when an existing configuration is copied, its parser configuration is also copied.

## Syntax

```
CREATE TEXT SEARCH CONFIGURATION name
(PARSER = parser_name | COPY = source_config)
[WITH ({configuration_option = value} [, ...])];
```

## Parameter Description

**Table 21-214** CREATE TEXT SEARCH CONFIGURATION parameters

| Parameter     | Description                                                                                      | Value Range                                                              |
|---------------|--------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| name          | Specifies the name (optionally schema-qualified) of the text search configuration to be created. | A string, which must comply with the <a href="#">naming convention</a> . |
| parser_name   | Specifies the name of the text search parser to use for this configuration.                      | -                                                                        |
| source_config | Specifies the name of an existing text search configuration to copy.                             | -                                                                        |

| Parameter            | Description                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| configuration_option | Specifies the configuration parameter of text search configuration is mainly for the parser executed by <b>parser_name</b> or contained by <b>source_config</b> . | Currently, the default, ngram, and zhparser parsers are supported. <ul style="list-style-type: none"><li>• The parser of the default type does not have the corresponding <b>configuration_option</b>.</li><li>• For details about the <b>configuration_option</b> of the ngram and zhparser parsers, see <a href="#">Table 21-215</a>.</li></ul> |

**Table 21-215** Configuration parameters for ngram and zhparser parsers

| Parser   | Parameters for adding an account | Description                                                                                                                 | Value Range                                                                                                                                                                                                                       |
|----------|----------------------------------|-----------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ngram    | gram_size                        | Length of word segmentation                                                                                                 | Integer, 1 to 4<br>Default value: 2                                                                                                                                                                                               |
|          | punctuation_ignore               | Whether to ignore punctuations                                                                                              | <ul style="list-style-type: none"><li>• <b>true</b> (default value): Ignore punctuations.</li><li>• <b>false</b>: Do not ignore punctuations.</li></ul>                                                                           |
|          | graphsymbol_ignore               | Whether to ignore graphical characters                                                                                      | <ul style="list-style-type: none"><li>• <b>true</b>: Ignore graphical characters.</li><li>• <b>false</b> (default value): Do not ignore graphical characters.</li></ul>                                                           |
| zhparser | punctuation_ignore               | Whether to ignore special characters including punctuations (\r and \n will not be ignored) in the word segmentation result | <ul style="list-style-type: none"><li>• <b>true</b> (default value): Ignore all the special characters including punctuations.</li><li>• <b>false</b>: Do not ignore all the special characters including punctuations.</li></ul> |
|          | seg_with_duality                 | Whether to aggregate segments with duality                                                                                  | <ul style="list-style-type: none"><li>• <b>true</b>: Aggregate segments with duality.</li><li>• <b>false</b> (default value): Do not aggregate segments with duality.</li></ul>                                                   |

| Parser | Parameters for adding an account | Description                                              | Value Range                                                                                                                                                                                                 |
|--------|----------------------------------|----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | multi_short                      | Whether to execute long words composite divide           | <ul style="list-style-type: none"><li>• <b>true</b> (default value): Execute long words composite divide.</li><li>• <b>false</b>: Do not execute long words composite divide.</li></ul>                     |
|        | multi_duality                    | Whether to aggregate segments in long words with duality | <ul style="list-style-type: none"><li>• <b>true</b>: Aggregate segments in long words with duality.</li><li>• <b>false</b> (default value): Do not aggregate segments in long words with duality.</li></ul> |
|        | multi_zmain                      | Whether to display key single words individually         | <ul style="list-style-type: none"><li>• <b>true</b>: Display key single words individually.</li><li>• <b>false</b> (default value): Do not display key single words individually.</li></ul>                 |
|        | multi_zall                       | Whether to display all single words individually.        | <ul style="list-style-type: none"><li>• <b>true</b>: Display all single words individually.</li><li>• <b>false</b> (default value): Do not display all single words individually.</li></ul>                 |

## Examples

Create a text search configuration named **ngram1** using the ngram parser.

```
CREATE TEXT SEARCH CONFIGURATION ngram1 (parser=ngram) WITH (gram_size = 2, grapsymbol_ignore = false);
```

Create a text search configuration named **ngram2** and copy all parameters of the existing configuration **ngram1**.

```
CREATE TEXT SEARCH CONFIGURATION ngram2 (copy=ngram1);
```

Create a text search configuration named **english\_1** and use the default text parser.

```
CREATE TEXT SEARCH CONFIGURATION english_1 (parser=default);
```

## Helpful Links

[ALTER TEXT SEARCH CONFIGURATION](#), [DROP TEXT SEARCH CONFIGURATION](#)

## 21.13.6.9 ALTER TEXT SEARCH CONFIGURATION

### Function

This syntax modifies the definition of a text search configuration.

Users can update the mapping from a token type to a dictionary, change the configuration's name or owner, and adjust the search configuration parameters.

### Precautions

- If a search configuration is referenced (to create an index), users are not allowed to modify it.
- To use **ALTER TEXT SEARCH CONFIGURATION**, you must be the owner of the configuration.

### Syntax

- Add a mapping from a token type to a dictionary.  

```
ALTER TEXT SEARCH CONFIGURATION name
 ADD MAPPING FOR token_type [, ...] WITH dictionary_name [, ...];
```
- Modify the text search configuration dictionary.  

```
ALTER TEXT SEARCH CONFIGURATION name
 ALTER MAPPING FOR token_type [, ...] REPLACE old_dictionary WITH new_dictionary;
```
- Modify the token type of the text search configuration.  

```
ALTER TEXT SEARCH CONFIGURATION name
 ALTER MAPPING FOR token_type [, ...] WITH dictionary_name [, ...];
```
- Change the text search configuration dictionary.  

```
ALTER TEXT SEARCH CONFIGURATION name
 ALTER MAPPING REPLACE old_dictionary WITH new_dictionary;
```
- Delete a mapping from a token type to a dictionary.  

```
ALTER TEXT SEARCH CONFIGURATION name
 DROP MAPPING [IF EXISTS] FOR token_type [, ...];
```
- Change the owner of the text search configuration.  

```
ALTER TEXT SEARCH CONFIGURATION name OWNER TO new_owner;
```
- Rename the text search configuration.  

```
ALTER TEXT SEARCH CONFIGURATION name RENAME TO new_name;
```
- Rename the text search configuration schema.  

```
ALTER TEXT SEARCH CONFIGURATION name SET SCHEMA new_schema;
```
- Modify the attributes of the text search configuration.  

```
ALTER TEXT SEARCH CONFIGURATION name SET ({ configuration_option = value } [, ...]);
```
- Reset the text search configuration attributes.  

```
ALTER TEXT SEARCH CONFIGURATION name RESET ({configuration_option} [, ...]);
```

 NOTE

- **ADD MAPPING FOR** adds a mapping for the token type to the text search configuration. If the mapping for any token type after **ADD MAPPING FOR** already exists in the text search configuration, the system reports an error.
- **ALTER MAPPING FOR** clears the existing token type mapping and then adds the specified token type mapping.
- **ALTER MAPPING REPLACE ... WITH ...** and **ALTER MAPPING FOR... REPLACE ... WITH ...** options replace **old\_dictionary** with **new\_dictionary**. Note that only when **pg\_ts\_config\_map** has tuples corresponding to **maptokentype** and **old\_dictionary**, the update will succeed. If the update fails, no messages are returned.
- **DROP MAPPING FOR** deletes the token type mapping specified in the current text search configuration. If **EXISTS** is not specified and the token type mapping specified by **DROP MAPPING FOR** does not exist in the text search configuration, the database reports an error.

## Parameter Description

**Table 21-216** ALTER TEXT SEARCH CONFIGURATION parameters

| Parameter       | Description                                                                                                                                                                   | Value Range                                                           |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| name            | Specifies the name of the text search configuration to be modified, which can include a schema.                                                                               | -                                                                     |
| token_type      | Specifies the name of the token type associated with the configured parser.                                                                                                   | For details, see <a href="#">Parsers</a> .                            |
| dictionary_name | Specifies the name of a text search dictionary to be consulted for the specified token types. If multiple dictionaries are listed, they are consulted in the specified order. | -                                                                     |
| old_dictionary  | Specifies the name of the text search dictionary to be replaced in the mapping.                                                                                               | -                                                                     |
| new_dictionary  | Specifies the name of a text search dictionary to be substituted for <b>old_dictionary</b> .                                                                                  | -                                                                     |
| new_owner       | Specifies the new owner of the text search configuration.                                                                                                                     | Valid username.                                                       |
| new_name        | Specifies the new name of the text search configuration.                                                                                                                      | A string compliant with the <a href="#">identifier naming rules</a> . |



| Parameter            | Description                                                     | Value Range                                                         |
|----------------------|-----------------------------------------------------------------|---------------------------------------------------------------------|
| new_schema           | Specifies the new schema name of the text search configuration. | Valid schema name.                                                  |
| configuration_option | Specifies the text search configuration item.                   | For details, see <a href="#">CREATE TEXT SEARCH CONFIGURATION</a> . |
| value                | Specifies the value of text search configuration option.        | -                                                                   |

## Examples

Create a text search configuration named **ngram1**.

```
DROP TEXT SEARCH CONFIGURATION IF EXISTS ngram1;
CREATE TEXT SEARCH CONFIGURATION ngram1 (parser=ngram) WITH (gram_size = 2, grapsymbol_ignore = false);
```

Add a mapping to the text search type **ngram1**.

```
ALTER TEXT SEARCH CONFIGURATION ngram1 ADD MAPPING FOR multisymbol WITH simple;
```

Change the owner of text search configuration.

```
CREATE USER joe password '{Password}';
ALTER TEXT SEARCH CONFIGURATION ngram1 OWNER TO joe;
```

Change the schema of text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION ngram1 SET SCHEMA joe;
```

Rename a text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION joe.ngram1 RENAME TO ngram_1;
```

Delete type mapping.

```
ALTER TEXT SEARCH CONFIGURATION joe.ngram_1 DROP MAPPING IF EXISTS FOR multisymbol;
```

Create a text search configuration.

```
DROP TEXT SEARCH CONFIGURATION IF EXISTS english_1;
CREATE TEXT SEARCH CONFIGURATION english_1 (parser=default);
```

Add a mapping to the text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR word WITH simple,english_stem;
```

Update the text search configuration by adding mappings.

```
ALTER TEXT SEARCH CONFIGURATION english_1 ADD MAPPING FOR email WITH english_stem,
french_stem;
```

Updating the text search configuration.

```
ALTER TEXT SEARCH CONFIGURATION english_1 ALTER MAPPING REPLACE french_stem with german_stem;
```

Query text search configurations.

```
SELECT b.cfgname,a.maptokentype,a.mapseqno,a.mapdict,c.dictname FROM pg_ts_config_map
a,pg_ts_config b, pg_ts_dict c WHERE a.mapcfg=b.oid AND a.mapdict=c.oid AND b.cfgname='english_1'
ORDER BY 1,2,3,4,5;
```

| cfgname   | maptokentype | mapseqno | mapdict | dictname     |
|-----------|--------------|----------|---------|--------------|
| english_1 | 2            | 1        | 3765    | simple       |
| english_1 | 2            | 2        | 14505   | english_stem |
| english_1 | 4            | 1        | 14505   | english_stem |
| english_1 | 4            | 2        | 14511   | german_stem  |
| (4 rows)  |              |          |         |              |

Links

[CREATE TEXT SEARCH CONFIGURATION](#), [DROP TEXT SEARCH CONFIGURATION](#)

21.13.6.10 DROP TEXT SEARCH CONFIGURATION

Function

**DROP TEXT SEARCH CONFIGURATION** deletes an existing text search configuration.

Precautions

To run the **DROP TEXT SEARCH CONFIGURATION** command, you must be the owner of the text search configuration.

Syntax

```
DROP TEXT SEARCH CONFIGURATION [IF EXISTS] name [CASCADE | RESTRICT];
```

Parameter Description

Table 21-217 DROP TEXT SEARCH CONFIGURATION parameters

| Parameter | Description                                                                                    | Value Range                                 |
|-----------|------------------------------------------------------------------------------------------------|---------------------------------------------|
| IF EXISTS | Sends a notice instead of an error if the specified text search configuration does not exist.  | -                                           |
| name      | Specifies the name (optionally schema-qualified) of a text search configuration to be deleted. | An existing text search configuration name. |

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                                                                                     |
|--------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• CASCADE: Automatically deletes objects that depend on the text search configuration to be deleted.</li><li>• RESTRICT: Refuses to delete the text search configuration if any objects depend on it. This is the default. This is the default.</li></ul> |

## Examples

Delete the text search configuration **ngram1**.

```
DROP TEXT SEARCH CONFIGURATION ngram1;
```

## Helpful Links

[ALTER TEXT SEARCH CONFIGURATION](#), [CREATE TEXT SEARCH CONFIGURATION](#)

### 21.13.6.11 CREATE TEXT SEARCH DICTIONARY

#### Function

**CREATE TEXT SEARCH DICTIONARY** creates a full-text retrieval dictionary. A full-text search dictionary is used to identify and process particular words during full-text retrieval.

Dictionaries are created by using predefined templates (defined in the **PG\_TS\_TEMPLATE** system catalog). Five types of dictionaries can be created, **Simple**, **Ispell**, **Synonym**, **Thesaurus**, and **Snowball**. Each type of dictionaries is used to handle different tasks.

#### Precautions

- A user with the **SYSADMIN** permission can create a dictionary. Then, the user automatically becomes the owner of the dictionary.
- A dictionary cannot be created in **pg\_temp** mode.
- After a dictionary is created or modified, any modification to the user-defined dictionary definition file will not affect the dictionary in the database. To make such modifications take effect in the dictionary in the database, run the **ALTER** statement to update the definition file of the dictionary.
- The **TEMPLATE** parameter in a dictionary cannot be modified.
- To specify a dictionary, specify both the dictionary definition file path (**FILEPATH**) and the file name (the parameter varies based on dictionary types).

- The name of a dictionary definition file can contain only lowercase letters, digits, and underscores (\_).
- The predefined dictionary file is stored in the **\$GAUSSHOME/share/postgresql/tsearch\_data** directory.

## Syntax

```
CREATE TEXT SEARCH DICTIONARY name (
 TEMPLATE = template
 [, option = value [, ...]]
);
```

## Parameter Description

**Table 21-218** CREATE TEXT SEARCH DICTIONARY parameters

| Parameter | Description                                                                                                                                                                                                                          | Value Range                                                                                                                                               |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| name      | Specifies the name (optionally schema-qualified) of the full-text search dictionary to be created.                                                                                                                                   | A string, which must comply with the <a href="#">naming convention</a> .                                                                                  |
| template  | Specifies a template name.                                                                                                                                                                                                           | Templates ( <b>Simple</b> , <b>Synonym</b> , <b>Thesaurus</b> , <b>Ispell</b> , and <b>Snowball</b> ) defined in the <b>PG_TS_TEMPLATE</b> system catalog |
| option    | Specifies the parameter name of a specific option for a full-text search dictionary. Each type of dictionaries has a template containing their custom parameters. Parameters function in a way irrelevant to their setting sequence. | For details, see <a href="#">Table 21-219</a> .                                                                                                           |
| value     | Specifies a parameter value. If the value is not an identifier or a number, enclose it with single quotation marks ("). You can also enclose identifiers and numbers with single quotation marks.                                    | -                                                                                                                                                         |

**Table 21-219** Option parameters of Simple, Ispell, Synonym, Thesaurus, and Snowball

| option                                    | Name of the option | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------------------------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parameters for a <b>Simple</b> dictionary | STOPWORD S         | Specifies the name of a file listing stop words. The default file name extension is .stop. For example, if the value of <b>STOPWORDS</b> is <b>french</b> , the actual file name is <b>french.stop</b> . In the file, each line defines a stop word. Dictionaries will ignore blank lines and spaces in the file and convert stop-word phrases into lowercase.                                                                                                |
|                                           | ACCEPT             | Specifies whether to accept a non-stop word as recognized. The default value is <b>true</b> .<br><br>If <b>ACCEPT=true</b> is set for a <b>Simple</b> dictionary, no token will be passed to subsequent dictionaries. In this case, you are advised to place the <b>Simple</b> dictionary at the end of the dictionary list. If <b>ACCEPT=false</b> is set, you are advised to place the <b>Simple</b> dictionary before at least one dictionary in the list. |

| option | Name of the option | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|        | FILEPATH           | <p>Specifies the directory for storing the stop word file. The stop word file can be stored locally or on the OBS server. If the file is stored locally, the directory format is '<b>file://absolute_path</b>'. If the file is stored on the OBS server, the directory format is '<b>obs://bucket/path accesskey=ak secretkey=sk region=region_name</b>'. The directory must be enclosed in single quotation marks (''). The default value is the directory where predefined dictionary files are located. Both the <b>FILEPATH</b> and <b>STOPWORDS</b> parameters need to be specified.</p> <p>To create a dictionary using the stop word file on the OBS server, perform the following steps:</p> <ol style="list-style-type: none"><li>1. Upload the stop word file to the OBS server. For example, upload the <b>french.stop</b> file to the <b>gaussdb</b> bucket on the OBS server <b>obsv3.sa-fb-1.externaldemo.com</b>. The URL is <b>https://gaussdb.obsv3.sa-fb-1.externaldemo.com/french.stop</b>. For details about how to upload the file and query the URL, see <i>Object Storage Service (OBS) User Guide</i>.</li><li>2. Add "<i>region_name</i>": "<i>obs domain</i>" to the <b>\$GAUSSHOME/etc/region_map</b> file. <i>region_name</i> can be a string consisting of uppercase letters, lowercase letters, digits, slashes (/), or underscores (_). <i>obs domain</i> indicates the domain name of the OBS server. For example, if <i>region_name</i> is set to <b>rg</b>, <b>region_map</b> is as follows: "<b>rg</b>": "<b>obsv3.sa-fb-1.externaldemo.com</b>".</li></ol> <p><b>NOTICE</b></p> <p><i>region_name</i> and <i>obs domain</i> are enclosed in double quotation marks. There is no space on the left of the colon and one space on the right of the colon.</p> <ol style="list-style-type: none"><li>3. Run the <b>CREATE TEXT SEARCH DICTIONARY</b> command to create a dictionary. The command is as follows:</li></ol> <pre>CREATE TEXT SEARCH DICTIONARY french_dict<br/>( TEMPLATE = pg_catalog.simple, STOPWORDS = french,<br/>FILEPATH = 'obs://gaussdb accesskey=xxx secretkey=yyy<br/>region=rg' );</pre> <p>The <b>french.stop</b> file is stored in the root directory of the <b>gaussdb</b> bucket. Therefore, the <i>path</i> is empty.</p> |

| option                                       | Name of the option | Description                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------------------------------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parameters for a <b>Synonym</b> dictionary   | SYNONYM            | Specifies the name of the definition file for a <b>Synonym</b> dictionary. The default file name extension is .syn.<br><br>The file is a list of synonyms. Each line is in the format of <i>token synonym</i> , that is, token and its synonym separated by a space.                                                                                                                  |
|                                              | CASESENSITIVE      | Specifies whether tokens and their synonyms are case sensitive. The default value is <b>false</b> , indicating that tokens and synonyms in dictionary files will be converted into lowercase. If this parameter is set to <b>true</b> , they will not be converted into lowercase.                                                                                                    |
|                                              | FILEPATH           | Specifies the directory for storing <b>Synonym</b> dictionary files. The directory can be a local directory or an OBS directory. The default value is the directory where predefined dictionary files are located. The directory format and the process of creating a <b>Synonym</b> dictionary using a file on the OBS server are the same as those of the <b>Simple</b> dictionary. |
| Parameters for a <b>Thesaurus</b> dictionary | DICTFILE           | Specifies the name of a dictionary definition file. The default file name extension is .ths.<br><br>The file is a list of synonyms. Each line is in the format of <i>sample words : indexed words</i> . The colon (:) is used as a separator between a phrase and its substitute word. If multiple sample words are matched, the TZ selects the longest one.                          |

| option                                     | Name of the option | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                            | DICTIONARY         | <p>Specifies the name of a subdictionary used for word normalization. This parameter is mandatory and only one subdictionary name can be specified. The specified subdictionary must exist. It is used to identify and normalize input text before phrase matching.</p> <p>If an input word cannot be recognized by the subdictionary, an error will be reported. In this case, remove the word or update the subdictionary to make the word recognizable. In addition, an asterisk (*) can be placed at the beginning of an indexed word to skip the application of a subdictionary on it, but all sample words must be recognizable by the subdictionary.</p> <p>If the sample words defined in the dictionary file contain stop words defined in the subdictionary, use question marks (?) to replace them. Assume that <b>a</b> and <b>the</b> are stop words defined in the subdictionary.</p> <p>? one ? two : swsw</p> <p><b>a one the two</b> and <b>the one a two</b> will be matched and output as <b>swsw</b>.</p> |
|                                            | FILEPATH           | <p>Specifies the directory for storing dictionary definition files. The directory can be a local directory or an OBS directory. The default value is the directory where predefined dictionary files are located. The directory format and the process of creating a <b>Synonym</b> dictionary using a file on the OBS server are the same as those of the <b>Simple</b> dictionary.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Parameters for an <b>Ispell</b> dictionary | DICTFILE           | <p>Specifies the name of a dictionary definition file. The default file name extension is .dict.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|                                            | AFFFILE            | <p>Specifies the name of an affix file. The default file name extension is .affix.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|                                            | STOPWORDS          | <p>Specifies the name of a file listing stop words. The default file name extension is .stop. The file content format is the same as that of the file for a <b>Simple</b> dictionary.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |



| option                                      | Name of the option | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|---------------------------------------------|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                             | FILEPATH           | Specifies the directory for storing dictionary files. The directory can be a local directory or an OBS directory. The default value is the directory where predefined dictionary files are located. The directory format and the process of creating a <b>Synonym</b> dictionary using a file on the OBS server are the same as those of the <b>Simple</b> dictionary.                                                                                            |
| Parameters for a <b>Snowball</b> dictionary | LANGUAGE           | Specifies the name of a language whose stemming algorithm will be used. According to spelling rules in the language, the algorithm normalizes the variants of an input word into a basic word or a stem.                                                                                                                                                                                                                                                          |
|                                             | STOPWORD S         | Specifies the name of a file listing stop words. The default file name extension is .stop. The file content format is the same as that of the file for a <b>Simple</b> dictionary.                                                                                                                                                                                                                                                                                |
|                                             | FILEPATH           | Specifies the directory for storing dictionary definition files. The directory can be a local directory or an OBS directory. The default value is the directory where predefined dictionary files are located. Both the <b>FILEPATH</b> and <b>STOPWORDS</b> parameters need to be specified. The directory format and the process of creating a <b>Snowball</b> dictionary using a file on the OBS server are the same as those of the <b>Simple</b> dictionary. |

## Examples

Create an **Ispell** dictionary **english\_ispell** (the dictionary definition file is from the open source dictionary).

```
CREATE TEXT SEARCH DICTIONARY english_ispell (
 TEMPLATE = ispell,
 DictFile = english,
 AffFile = english,
 StopWords = english,
 FilePath = 'obs://bucket_name/path accesskey=ak secretkey=sk region=rg'
);
```

See examples in [Configuration Examples](#).

## Helpful Links

[ALTER TEXT SEARCH DICTIONARY](#), [CREATE TEXT SEARCH DICTIONARY](#)

## 21.13.6.12 ALTER TEXT SEARCH DICTIONARY

### Function

Modifies the definition of a full-text retrieval dictionary, including its parameters, name, owner, and schema.

### Precautions

- **ALTER** is not supported by predefined dictionaries.
- Only the owner of a dictionary can do **ALTER** to the dictionary. System administrators have this permission by default.
- After a dictionary is created or modified, any modification to the user-defined dictionary definition file in the directory specified by **FilePath** will not affect the dictionary in the database. To make such modifications take effect in the dictionary in the database, run the **ALTER TEXT SEARCH DICTIONARY** statement to update the definition file of the dictionary.

### Syntax

- Modify the dictionary definition.  
`ALTER TEXT SEARCH DICTIONARY name ( option = value | option [, ...] );`
- Rename a dictionary.  
`ALTER TEXT SEARCH DICTIONARY name RENAME TO new_name;`
- Set the schema of a dictionary.  
`ALTER TEXT SEARCH DICTIONARY name SET SCHEMA new_schema;`
- Change the owner of a dictionary.  
`ALTER TEXT SEARCH DICTIONARY name OWNER TO new_owner;`

### Parameter Description

Table 21-220 ALTER TEXT SEARCH DICTIONARY parameters

| Parameter | Description                                                                                       | Value Range                                  |
|-----------|---------------------------------------------------------------------------------------------------|----------------------------------------------|
| name      | Specifies the name of the full-text search dictionary to be modified, which can include a schema. | Name of a valid full-text search dictionary. |

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Value Range                                                                                         |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
| option     | <p>Specifies the parameter name of the full-text search dictionary specific option to be modified. Each type of dictionaries has a template containing their custom parameters. Parameters function in a way irrelevant to their setting sequence.</p> <ul style="list-style-type: none"><li>• The <b>TEMPLATE</b> parameter in a dictionary cannot be modified.</li><li>• To specify a dictionary, specify both the dictionary definition file path (<b>FILEPATH</b>) and the file name (the parameter varies based on dictionary types).</li><li>• The name of a dictionary definition file can contain only lowercase letters, digits, and underscores (_).</li></ul> | For details about the parameters, see the <b>option</b> parameter in <a href="#">Table 21-218</a> . |
| value      | Specifies the new value of a parameter. If = and <i>value</i> are omitted, the previous settings of the parameter will be deleted and the default value will be used.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Corresponding <b>option</b> definition.                                                             |
| new_name   | Specifies the new name of the full-text search dictionary.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | A string compliant with the <a href="#">identifier naming rules</a> .                               |
| new_owner  | Specifies the new owner of the full-text search dictionary.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Valid username.                                                                                     |
| new_schema | Specifies the new schema of the full-text search dictionary.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Valid schema name.                                                                                  |

## Examples

Create a full-text search dictionary **my\_dict**.

```
CREATE TEXT SEARCH DICTIONARY my_dict (
 TEMPLATE = snowball,
 Language = english,
 StopWords = english
);
```

Change the **Language** setting for the full-text search dictionary **my\_dict** and remove the stop word definition.

```
ALTER TEXT SEARCH DICTIONARY my_dict (Language = dutch,StopWords);
```

Update the definition of the full-text search dictionary **my\_dict** without changing any content.

```
ALTER TEXT SEARCH DICTIONARY my_dict (dummy);
```

## Helpful Links

[CREATE TEXT SEARCH DICTIONARY](#) and [DROP TEXT SEARCH DICTIONARY](#)

### 21.13.6.13 DROP TEXT SEARCH DICTIONARY

#### Function

**DROP TEXT SEARCH DICTIONARY** deletes a full-text retrieval dictionary.

#### Precautions

- **DROP** is not supported by predefined dictionaries.
- Only the owner of a dictionary can do **DROP** to the dictionary. System administrators have this permission by default.
- Execute **DROP...CASCADE** only when necessary because this operation will delete the text search configuration that uses this dictionary.

#### Syntax

```
DROP TEXT SEARCH DICTIONARY [IF EXISTS] name [CASCADE | RESTRICT]
```

#### Parameter Description

**Table 21-221** DROP TEXT TEXT SEARCH DICTIONARY parameters

| Parameter | Description                                                                                                                                               | Value Range                     |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| IF EXISTS | Sends a notice instead of an error if the specified full-text search dictionary does not exist.                                                           | -                               |
| name      | Specifies the name of a dictionary to be deleted. (If you do not specify a schema name, the dictionary in the current schema will be deleted by default.) | Name of an existing dictionary. |

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: Automatically deletes dependent objects of a dictionary and then deletes all dependent objects of these objects in sequence. If any text search configuration that uses the dictionary exists, <b>DROP</b> execution will fail. You can add <b>CASCADE</b> to delete all text search configurations and dictionaries that use the dictionary.</li><li>• <b>RESTRICT</b>: Rejects the deletion of a dictionary if any object depends on the dictionary. This is the default.</li></ul> |

## Examples

Delete **english** dictionary.

```
DROP TEXT SEARCH DICTIONARY english;
```

## Helpful Links

[ALTER TEXT SEARCH DICTIONARY](#), [CREATE TEXT SEARCH DICTIONARY](#)

### 21.13.6.14 CLUSTER

#### Function

This syntax clusters tables by their indexes.

- **CLUSTER** clusters the table specified by table name based on the index specified by index name. The index specified by index name must have been defined in the specified table.
- When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation: when the table is subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order.
- When a table is clustered, DWS records which index the table was clustered by. The form **CLUSTER table\_name** reclusters the table using the same index used earlier. You can also use **SET WITHOUT CLUSTER** of **CLUSTER** or **ALTER**

**TABLE** to set indexes for subsequent clustering operations or clear any previous settings.

- **CLUSTER** without any parameters reclusters all tables that have been clustered in the database owned by the current user. If **CLUSTER** is invoked by the system administrator, it indicates all tables that have been clustered.
- When a table is being clustered, an **ACCESS EXCLUSIVE** lock is acquired on it. This prevents any other database operations (both reads and writes) from operating on the table until the clustering is complete.

## Precautions

- **CLUSTER** cannot be executed in transactions.
- **CLUSTER** cannot be performed on HDFS tables.
- **CLUSTER** cannot be performed on global temporary tables.
- Performing a **CLUSTER** operation on a table triggers a rebuilding process where data is dumped into a new data file. Once the process is complete, the original file is deleted. Keep in mind that if the table is large, this process can consume a significant amount of disk space. When the disk space is insufficient, exercise caution when performing the **CLUSTER** operation on large tables to prevent the cluster from being read-only.
- In the storage-compute decoupled architecture, a message is displayed indicating that operations on the entire cluster database are not supported.
- Only row-store B-tree indexes support **CLUSTER**.
- In cases where you are accessing single rows randomly within a table, the actual order of the data in the table is unimportant. However, if you tend to access some data more than others, and there is an index that groups them together, you will benefit from using **CLUSTER**. If you request a range of index values from a table or if an index value matches multiple rows, **CLUSTER** helps. It does this by storing related rows on the same page. This saves disk access time and speeds up queries.
- During clustering, the system creates a temporary copy of the table in the index sequence and a temporary copy of each index in the table. The disk must have enough space for both the table and its index.
- **CLUSTER** stores clustering information. You can manually cluster tables initially, then set up a periodic maintenance script. This allows tables to be automatically clustered regularly.
- Because the optimizer records statistics about the ordering of tables, it is advisable to run **ANALYZE** on the newly clustered table. Otherwise, the optimizer might select an incorrect query plan.

## Syntax

- Cluster a table.  
`CLUSTER [ VERBOSE ] table_name [ USING index_name ];`
- Cluster a partition.  
`CLUSTER [ VERBOSE ] table_name PARTITION ( partition_name ) [ USING index_name ];`
- Cluster the table that has previously been clustered.  
`CLUSTER [ VERBOSE ];`

## Parameter Description

**Table 21-222** CLUSTER parameters

| Parameter      | Description                               | Value Range                    |
|----------------|-------------------------------------------|--------------------------------|
| VERBOSE        | Enables the display of progress messages. | -                              |
| table_name     | Specifies the name of the table.          | Name of an existing table.     |
| index_name     | Specifies the index name.                 | Name of an existing index.     |
| partition_name | Specifies the partition name.             | Name of an existing partition. |

## Examples

Create a partitioned table.

```
CREATE TABLE tpcds.inventory_p1
(
 INV_DATE_SK INTEGER NOT NULL,
 INV_ITEM_SK INTEGER NOT NULL,
 INV_WAREHOUSE_SK INTEGER NOT NULL,
 INV_QUANTITY_ON_HAND INTEGER
)
DISTRIBUTE BY HASH(INV_ITEM_SK)
PARTITION BY RANGE(INV_DATE_SK)
(
 PARTITION P1 VALUES LESS THAN(2451179),
 PARTITION P2 VALUES LESS THAN(2451544),
 PARTITION P3 VALUES LESS THAN(2451910),
 PARTITION P4 VALUES LESS THAN(2452275),
 PARTITION P5 VALUES LESS THAN(2452640),
 PARTITION P6 VALUES LESS THAN(2453005),
 PARTITION P7 VALUES LESS THAN(MAXVALUE)
);
```

Create an index named **ds\_inventory\_p1\_index1**.

```
CREATE INDEX ds_inventory_p1_index1 ON tpcds.inventory_p1 (INV_ITEM_SK) LOCAL;
```

Cluster the **tpcds.inventory\_p1** table.

```
CLUSTER tpcds.inventory_p1 USING ds_inventory_p1_index1;
```

Cluster the **p3** partition.

```
CLUSTER tpcds.inventory_p1 PARTITION (p3) USING ds_inventory_p1_index1;
```

Cluster the tables that can be clustered in the database.

```
CLUSTER;
```

## 21.13.7 Stored Procedures, Functions, Triggers, and Cursors

## 21.13.7.1 CREATE FUNCTION

### Function

A function is a reusable code block that accepts input parameters, performs specific operations, and returns a value or result set. **CREATE FUNCTION** is used to create a user-defined function (UDF) in a database, enabling other computing functions that are not supported by the built-in **system functions** of DWS and covering complex computing scenarios specific to users.

The functions have the following features:

- They support input parameters and return results depending on specific conditions.
- You must declare the return value's data type. It outputs either a single value (scalar function) or a table (table-valued function).
- By wrapping complex logic into one function, you can call it like a built-in SQL function, enhancing code reuse.

### Three DWS Function Types Based on Behaviors

SQL functions fall into three types based on their behaviors: immutable, stable, and volatile. **Table 21-223** shows their relationships.

**Table 21-223** Function types

| Function Type | Definition                                                                                                                                                                                                                                                                                                         | Typical Use Case                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| immutable     | An immutable function's output depends solely on its input parameters, ignoring external factors like database state, time, or transaction context. It consistently produces the same result when given identical inputs, regardless of when or where it is called. Additionally, it does not alter database data. | Mathematical calculations, hashing, and string operations—examples include ABS(x), POWER(a, b), and MD5(str).           |
| stable        | A stable function produces consistent results within a single query if the inputs remain unchanged. Its output depends on the database's current state, like table data or session variables, so results may vary across separate queries or transactions.                                                         | Tasks like filtering or aggregating live data, such as using MAX(col) or MIN(col), which stay constant during one scan. |



| Function Type | Definition                                                                                                                                                                                                                                                | Typical Use Case                                                                                                                                                                    |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| volatile      | Volatile functions are highly unpredictable. Their results can differ with every call or alter the database state by performing actions like writing data. Since their behavior cannot be predicted, the optimizer must recompute them on each execution. | Tasks like generating random numbers, retrieving timestamps, or executing DML operations. For instance, calling the <code>RANDOM()</code> function produces a new value every time. |

You can view the **provolatile** column in the **PG\_PROC** system catalog to check the function type. (**i** indicates immutable, **s** indicates stable, and **v** indicates volatile.)

## Pushing Functions Down to DNs (SHIPPABLE and NOT SHIPPABLE)

The **CREATE FUNCTION** syntax uses **SHIPPABLE** and **NOT SHIPPABLE** to determine if the function runs on the data node (DN).

- Functions of the **IMMUTABLE** type can always be pushed down to the DNs.
- Functions of the **STABLE** or **VOLATILE** type can be pushed down to DNs only if their attribute is **SHIPPABLE**.

Exercise caution when defining the **SHIPPABLE** attribute for a function. **SHIPPABLE** means that the entire function will be pushed down to DNs for execution. If the attribute is incorrectly set, serious problems such as result errors may occur.

The **SHIPPABLE** attribute, like **IMMUTABLE**, comes with usage limits. It requires functions to exclude elements that prevent pushdown. Once pushed down to a DN, the function's calculations must rely solely on the DN's dataset.

### Functions cannot be marked as IMMUTABLE if they:

1. Reference tables or views because their results may change when the data updates.
2. Call other **STABLE** or **VOLATILE** functions.
3. Include non-pushdown components, which clash with the **IMMUTABLE** rule. This often involves unsupported pushdown functions or syntax.
4. Perform aggregations needing **STREAM** plans (such as the `listagg` function), as these require computations on both DNs and CNs.

### Functions cannot be marked as SHIPPABLE if they:

1. Reference a hash-distributed table.
2. Include non-pushdown components, functions, or syntax. For details, see "Optimizing Statement Pushdown".
3. Involve calculations with data spread across DNs, such as aggregations.

 **CAUTION**

To optimize performance DWS, you need to define function attributes accurately. Incorrect settings can reduce query speed and produce inconsistent results.

## Precautions

- The precision values (if any) of the parameters or return values of a function are not checked.
- Always define a clear schema for table operations in your functions. For instance, use **hr.table1** to point to a table in the **hr** schema. Without this, the function might not run correctly.
- **current\_schema** and **search\_path** specified by **SET** during function creation are invalid. The values of **search\_path** and **current\_schema** after function execution are the same as those before function execution.
- If a function has output parameters, the **SELECT** statement uses the default values of the output parameters when calling the function. When the **CALL** statement calls the function, it requires that the output parameter values are adapted to Oracle. When the **CALL** statement calls an overloaded PACKAGE function, it can use the default values of the output parameters. For details, see examples in [CALL](#).
- Only the functions compatible with PostgreSQL or those with the **PACKAGE** attribute can be overloaded. After **REPLACE** is specified, a new function is created instead of replacing a function if the number of parameters, parameter type, or return value is different.
- You can use the **SELECT** statement to specify different parameters using identical functions, but cannot use the **CALL** statement to call identical functions without the **PACKAGE** attribute because **CALL** aligns with Oracle syntax.
- When you create a function, you cannot insert other agg functions out of the avg function or other functions.
- In non-logical cluster mode, return values, parameters, and variables cannot be set to the tables of the Node Groups that are not installed in the system by default. The internal statements of SQL functions cannot be executed on such tables.
- In logical cluster mode, if return values and parameters of the function are user tables, all the tables must be in the same logical cluster. If the function body involves operations on multiple logical cluster tables, the function cannot be set to **IMMUTABLE** or **SHIPPABLE**, preventing the function from being pushed down to a DN.
- In logical cluster mode, the parameters and return values of the function cannot use the **%type** to reference a table column type. Otherwise, the function will fail to be created.
- By default, the permissions to execute new functions are granted to **PUBLIC**. For details, see [GRANT](#). You can revoke the default execution permissions from **PUBLIC** and grant them to other users as needed. To avoid the time window during which new functions can be accessed by all users, create functions in transactions and set function execution permissions.

- In a cluster with multiple CNs, the input or output parameters of a function cannot be set to the temporary table type. This is because the correct temporary schema cannot be obtained based on the table name when the function is created on a non-current CN. As a result, the accurate table type cannot be obtained.

## Syntax

- Syntax (compatible with PostgreSQL) for creating a user-defined function:

```
CREATE [OR REPLACE] FUNCTION function_name
([{ argname [argmode] argtype [{ DEFAULT | := | = } expression] } [, ...]])
[RETURNS rettype [DETERMINISTIC] | RETURNS TABLE ({ column_name column_type }
[, ...])]
LANGUAGE lang_name
[
 { IMMUTABLE | STABLE | VOLATILE }
 | { SHIPPABLE | NOT SHIPPABLE }
 | WINDOW
 | [NOT] LEAKPROOF
 | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
 | { [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER | AUTHID DEFINER |
AUTHID CURRENT_USER }
 | { FENCED | NOT FENCED }
 | { PACKAGE }

 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } value | FROM CURRENT } }
][...]
{
 AS 'definition'
 | AS 'obj_file', 'link_symbol'
}
```

- Oracle syntax of creating a customized function:

```
CREATE [OR REPLACE] FUNCTION function_name
([{ argname [argmode] argtype [{ DEFAULT | := | = } expression] } [, ...]])
RETURN rettype [DETERMINISTIC]
[
 { IMMUTABLE | STABLE | VOLATILE }
 | { SHIPPABLE | NOT SHIPPABLE }
 | { PACKAGE }
 | { FENCED | NOT FENCED }
 | [NOT] LEAKPROOF
 | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
 | { [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER |
AUTHID DEFINER | AUTHID CURRENT_USER }
]
[
 | COST execution_cost
 | ROWS result_rows
 | SET configuration_parameter { { TO | = } value | FROM CURRENT }
]
][...]
{
 IS | AS
} plsql_body
/
```

## Parameter Description

**Table 21-224** CREATE FUNCTION parameters

| Parameter     | Description                                                                                                                                                                                                                                                                                                                            | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| function_name | Specifies the name of the function to be created.<br><b>CAUTION</b><br>If the name of the function to be created is the same as that of a system function, you are advised to specify a schema. When invoking a user-defined function, you need to specify a schema. Otherwise, the system preferentially invokes the system function. | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                                                                                                                                                                            |
| argname       | Specifies the name of a function parameter.                                                                                                                                                                                                                                                                                            | A string, which must comply with the naming convention.                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| argmode       | Specifies the mode of a function parameter.                                                                                                                                                                                                                                                                                            | <b>IN, OUT, INOUT, or VARIADIC.</b> <ul style="list-style-type: none"><li>• <b>IN</b>: input parameter. The default value is <b>IN</b>.</li><li>• <b>OUT</b>: output parameter. Only this type can include <b>VARIADIC</b>.</li><li>• <b>INOUT</b>: both input and output parameters.</li><li>• <b>OUT VARIADIC</b>: output parameter, which is defined as an array type.</li></ul> <b>CAUTION</b><br>The parameters of <b>OUT</b> and <b>INOUT</b> cannot be used in function definition of <b>RETURNS TABLE</b> . |
| argtype       | Data types of function parameters.                                                                                                                                                                                                                                                                                                     | For details about common data types, see <a href="#">Data Types</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| expression    | Indicates the default expression of a parameter.                                                                                                                                                                                                                                                                                       | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

| Parameter          | Description                                                                                                                                                                                                                                                                                          | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rettype            | Indicates the return data type.                                                                                                                                                                                                                                                                      | When there is <b>OUT</b> or <b>IN OUT</b> parameter, the <b>RETURNS</b> clause can be omitted. If the clause exists, the result type of the clause must be the same as that of the output parameter. If there are multiple output parameters, the result type of the clause is <b>RECORD</b> . Otherwise, the result type of the clause is the same as that of a single output parameter.<br><br>The <b>SETOF</b> modifier indicates that the function will return a set of items, rather than a single item. |
| DETERMINISTIC      | Adapted to Oracle SQL syntax. You are not advised to use it.                                                                                                                                                                                                                                         | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| column_name        | Specifies the column name.                                                                                                                                                                                                                                                                           | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| column_type        | Specifies the column type.                                                                                                                                                                                                                                                                           | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| definition         | Specifies a string constant defining the function; the meaning depends on the language. It can be an internal function name, a path pointing to a target file, a SQL query, or text in a procedural language.                                                                                        | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| LANGUAGE lang_name | Indicates the name of the language that is used to implement the function. It can be <b>SQL</b> , <b>internal</b> , or the name of user-defined process language. To ensure downward compatibility, the name can use single quotation marks. Contents in single quotation marks must be capitalized. | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

| Parameter | Description                                                                                                                                                                                                                                                                                                                       | Value Range |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| WINDOW    | <p>Indicates that the function is a window function. The WINDOW attribute cannot be changed when the function definition is replaced.</p> <p><b>CAUTION</b><br/>For a user-defined window function, the value of <b>LANGUAGE</b> can only be <b>internal</b>, and the referenced internal function must be a window function.</p> | -           |

| Parameter | Description                                                                                      | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------|--------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IMMUTABLE | Indicates that the function always returns the same result if the parameter values are the same. | <p>If the input argument of the function is a constant, the function value is calculated at the optimizer stage. The advantage is that the expression value can be obtained as early as possible, so the cost estimation is more accurate and the execution plan generated is better.</p> <p>A user-defined <b>IMMUTABLE</b> function is automatically pushed down to DN for execution, which may cause potential risks. If a function is defined as <b>IMMUTABLE</b> but the function execution process is in fact not <b>IMMUTABLE</b>, serious problems such as result errors may occur. Therefore, exercise caution when defining the <b>IMMUTABLE</b> attribute for a function.</p> <p>Examples:</p> <ol style="list-style-type: none"><li>1. If a user-defined function references objects such as tables and views, the function cannot be defined as <b>IMMUTABLE</b>, because the function may return different results when the data in a referenced table changes.</li><li>2. Call other <b>STABLE</b> or <b>VOLATILE</b> functions.</li><li>3. Include non-pushdown components, which clash with the <b>IMMUTABLE</b> rule. Typical scenarios include functions and syntax that cannot be pushed down.</li><li>4. If a user-defined function contains an aggregation operation that will generate <b>STREAM</b> plans to complete the operation (meaning that DN and CN are involved for results calculation, such as the <b>LISTAGG</b> function), the function cannot be defined as <b>IMMUTABLE</b>.</li></ol> |

| Parameter | Description                                                                                                                                                                                                          | Value Range                                                                                                                                                                                                                              |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           |                                                                                                                                                                                                                      | To prevent possible problems, you can set <b>behavior_compat_options</b> to <b>check_function_conflicts</b> in the database to check definition conflicts. This method can identify the <b>1</b> and <b>2</b> scenarios described above. |
| STABLE    | Indicates that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter values, but that its result varies by SQL statements. | -                                                                                                                                                                                                                                        |
| VOLATILE  | Indicates that the function value can change even within a single table scan, so no optimizations can be made.                                                                                                       | -                                                                                                                                                                                                                                        |



| Parameter                  | Description                                                             | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------|-------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SHIPPABLE<br>NOT SHIPPABLE | Indicates whether the function can be pushed down to DNs for execution. | <ul style="list-style-type: none"><li>Functions of the IMMUTABLE type can always be pushed down to the DNs.</li><li>Functions of the STABLE or VOLATILE type can be pushed down to DNs only if their attribute is <b>SHIPPABLE</b>. Exercise caution when defining the <b>SHIPPABLE</b> attribute for a function. <b>SHIPPABLE</b> means that the entire function will be pushed down to DNs for execution. If the attribute is incorrectly set, serious problems such as result errors may occur.</li></ul> <p>Similar to the <b>IMMUTABLE</b> attribute, the <b>SHIPPABLE</b> attribute has use restrictions. The function cannot contain factors that do not allow the function to be pushed down for execution. If a function is pushed down to a single DN for execution, the function's calculation logic will depend only on the data set of the DN.</p> <p>Examples:</p> <ol style="list-style-type: none"><li>If a function references a hash table, you cannot define the function as <b>SHIPPABLE</b>.</li><li>If a function contains factors, functions, or syntax that cannot be pushed down, the function cannot be defined as SHIPPABLE. For details, see Optimizing Statement Pushdown.</li><li>If a function's calculation process involves data across DNs, the function cannot be defined as <b>SHIPPABLE</b>. For example, some aggregation operations involve data across DNs.</li></ol> |

| Parameter                            | Description                                                                                                                                                                                                                                                                                                                                                      | Value Range                                                                                                                                                                                                                                                            |
|--------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PACKAGE                              | Indicates whether the function can be overloaded. PostgreSQL-style functions can be overloaded, and this parameter is designed for Oracle-style functions.                                                                                                                                                                                                       | <ul style="list-style-type: none"><li>• All PACKAGE and non-PACKAGE functions cannot be overloaded or replaced.</li><li>• PACKAGE functions do not support parameters of the VARIADIC type.</li><li>• The PACKAGE attribute of functions cannot be modified.</li></ul> |
| LEAKPROOF                            | Indicates that the function has no side effects. <b>LEAKPROOF</b> can be set only by the system administrator.                                                                                                                                                                                                                                                   | -                                                                                                                                                                                                                                                                      |
| CALLED ON NULL INPUT                 | Declares that some parameters of the function can be invoked in normal mode if the parameter values are <b>NULL</b> . This parameter can be omitted.                                                                                                                                                                                                             | -                                                                                                                                                                                                                                                                      |
| RETURNS NULL ON NULL INPUT<br>STRICT | Indicates that the function always returns <b>NULL</b> whenever any of its parameters are <b>NULL</b> . If this parameter is specified, the function is not executed when there are <b>NULL</b> parameters; instead a <b>NULL</b> result is returned automatically.<br><br>The usage of <b>RETURNS NULL ON NULL INPUT</b> is the same as that of <b>STRICT</b> . | -                                                                                                                                                                                                                                                                      |
| EXTERNAL                             | The keyword EXTERNAL is allowed for SQL conformance, but it is optional since, unlike in SQL, this feature applies to all functions not only external ones.                                                                                                                                                                                                      | -                                                                                                                                                                                                                                                                      |

| Parameter                               | Description                                                                                                                                                                                                                                                         | Value Range                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SECURITY INVOKER<br>AUTHID CURRENT_USER | Indicates that the function is to be executed with the permissions of the user that calls it. This parameter can be omitted.<br><br><b>SECURITY INVOKER</b> and <b>AUTHID CURRENT_USER</b> have the same functions.                                                 | -                                                                                                                                                                                                                                                                                                                                                                                                                |
| SECURITY DEFINER<br>AUTHID DEFINER      | Specifies that the function is to be executed with the permissions of the user that created it.<br><br>The usage of <b>AUTHID DEFINER</b> is the same as that of <b>SECURITY DEFINER</b> .                                                                          | -                                                                                                                                                                                                                                                                                                                                                                                                                |
| FENCED<br>NOT FENCED                    | (Effective only for C functions) Specifies whether functions are executed in fenced mode. In NOT FENCED mode, a function is executed in a CN or DN process. In FENCED mode, a function is executed in a new fork process, which does not affect CN or DN processes. | Application scenarios: <ul style="list-style-type: none"><li>• Develop or debug a function in FENCED mode and execute it in NOT FENCED mode. This reduces the cost of the fork process and communication.</li><li>• Perform complex OS operations, such as open a file, process signals and threads, in FENCED mode so that DWS running is not affected.</li><li>• The default value is <b>FENCED</b>.</li></ul> |
| COST<br>execution_cost                  | A positive number giving the estimated execution cost for the function. The unit of <b>execution_cost</b> is <b>cpu_operator_cost</b> .                                                                                                                             | Positive                                                                                                                                                                                                                                                                                                                                                                                                         |
| ROWS<br>result_rows                     | Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.                                                                                                                                          | Its value is a positive number. The default value is <b>1000</b> .                                                                                                                                                                                                                                                                                                                                               |

| Parameter               | Description                                                                                                                                                      | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| configuration_parameter | Specifies the parameter value configured for the function.                                                                                                       | <ul style="list-style-type: none"><li>• <b>value</b><br/>Sets a specified database session parameter to a specified value. If the value is <b>DEFAULT</b> or <b>RESET</b>, the default setting is used in the new session. <b>OFF</b> closes the setting.<br/>Value range: a string<ul style="list-style-type: none"><li>– DEFAULT</li><li>– OFF</li><li>– RESET</li></ul>Specifies the default value.</li><li>• <b>from current</b><br/>Uses the value of <b>configuration_parameter</b> of the current session.</li></ul> |
| plsql_body              | Indicates the PL/SQL stored procedure body.<br><br>When the function is creating users, the log will record unencrypted passwords. You are not advised to do it. | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## Example: Calculate the Sum of Two Integers Using an SQL Function

Syntax:

```
CREATE OR REPLACE FUNCTION func_add_sql(
integer, --The first integer parameter is not named and is referenced by the location parameter
$1.
integer --The second integer parameter is not named and is referenced by the location
parameter $2.
)
RETURNS integer --The return type of the function is integer.
AS 'select $1 + $2;' --The function body adds the location parameters $1 and $2.
LANGUAGE SQL --The SQL language is used to implement this function.
IMMUTABLE --The function type is IMMUTABLE, which indicates that the same result is
returned for the same input.
RETURNS NULL ON NULL INPUT; --It is same as STRICT. If any parameter is NULL, the function
returns NULL.
```

Return the sum of the two input parameters.

```
SELECT func_add_sql(8,3);
```

func\_add\_sql

11

Return null because the input parameter is null.

```
SELECT func_add_sql(Null,3);
```

`func_add_sql`

*(Null)*

## Example: Increment an Integer, in PL/pgSQL

Syntax:

```
CREATE OR REPLACE FUNCTION func_increment_plsql(
i integer --The input parameter is of the integer type.
) RETURNS integer --The return value of the function is of the integer type.
AS $$
 BEGIN --The function body begins.
 RETURN i + 1; --The return result is the sum of 1 and the input parameter.
 END;
$$ LANGUAGE plpgsql; --The function is written in PL/pgSQL.
```

Return the integer auto-increment result.

```
SELECT func_increment_plsql(5);
```

`func_increment_plsql`

6

The input parameter is specified by the first column of the **numbers** table. After the function is used for calculation, the calculation result is returned to the second column.

```
DROP TABLE IF EXISTS numbers;
CREATE TABLE numbers (value integer);
INSERT INTO numbers VALUES (1), (2), (3), (4), (5);
SELECT value, func_increment_plsql(value) AS incremented_value FROM numbers ORDER BY 1;
```

| value | incremented_value |
|-------|-------------------|
| 1     | 2                 |
| 2     | 3                 |
| 3     | 4                 |
| 4     | 5                 |
| 5     | 6                 |

## Example: Return a Record in PL/pgSQL

```
CREATE OR REPLACE FUNCTION compute(
i int, ---The input parameter is of the integer type.
out result_1 bigint, --The output result is 1 and the data type is large integer.
out result_2 bigint --The output result is 2 and the data type is large integer.
)
RETURNS SETOF RECORD --A record containing more than one output parameter is returned.
AS $$
BEGIN
 result_1 = i + 1; -- It is used to calculate the first result value. The input value is increased by 1.
 result_2 = i * 10; -- It is used to calculate the second result value. The input value is multiplied
by 10.
 RETURN next; --It returns the current calculation result and then continue to execution
(multiple rows can be returned).
END;
$$LANGUAGE plpgsql; --The function is written in PL/pgSQL.
```

Return the following information when the function is called once.

```
SELECT * FROM compute(5);
```

| result_1 | result_2 |
|----------|----------|
| 6        | 50       |

Generate a sequence of 1, 2, and 3, and call the compute function for each value in the sequence.

```
SELECT i, (compute(i)).* --Call the compute function first, and then expand all columns returned by the
function.
FROM generate_series(1, 3) --Generate a temporary sequence containing 1, 2, and 3 to assign values to
i.
AS t(i); --Assign the result to temporary table t, in which i is the column name.
```

| i | result_1 | result_2 |
|---|----------|----------|
| 1 | 2        | 10       |
| 2 | 3        | 20       |
| 3 | 4        | 30       |

The data in table **test\_data** is used. The **value** column in the table is used as the input parameter of the compute function. The calculation result is as follows:

```
DROP TABLE IF EXISTS test_data;
CREATE TABLE test_data (id serial, value int);
INSERT INTO test_data (value) VALUES (2), (4), (6);
```

```
SELECT id, value, (compute(value)).*
FROM test_data ORDER BY 1;
```

| id | value | result_1 | result_2 |
|----|-------|----------|----------|
| 1  | 2     | 3        | 20       |
| 2  | 4     | 5        | 40       |
| 3  | 6     | 7        | 60       |

## Example: Create an SQL Function that Returns Multiple Output Parameters

Syntax:

```
CREATE OR REPLACE FUNCTION func_dup_sql(
in int, --The input parameter is of the integer type.
out f1 int, --The first output parameter is of the integer type.
out f2 text) --The second output parameter is of the text type.
AS $$
SELECT $1, CAST($1 AS text) || 's text' -- Function body. The query returns two columns. The first column is
the input parameter $1, and the second column is the parameter converted to text and concatenated with
is text.
$$ LANGUAGE SQL;
```

Execute the function.

```
SELECT * FROM func_dup_sql(42);
```

| f1 | f2         |
|----|------------|
| 42 | 42 is text |

## Example: Create an Overloaded Function with the PACKAGE Attribute

Define two overloaded functions named **package\_func\_overload** in **package**. The two functions have different meanings depending on the parameter type.

```
CREATE OR REPLACE FUNCTION package_func_overload(
col int, --The first input parameter is of the integer type.
col2 int --The second input parameter is of the integer type.
)
RETURN integer PACKAGE
AS
DECLARE
 col_type text; -- Declare a local variable.
BEGIN
 col := 122; --Set the first parameter to 122.
 dbms_output.put_line('two int parameters ' || col2);
 RETURN 0; --Return the integer 0.
END;
/

CREATE OR REPLACE FUNCTION package_func_overload(
col int,
col2 smallint
)
RETURN integer PACKAGE
AS
DECLARE
 col_type text;
BEGIN
 col := 122;
 dbms_output.put_line('two smallint parameters ' || col2);
 RETURN 0;
END;
/
```

## Function Pushdown Logic and Example

Pushdown logic for functions has two main aspects:

1. The system checks if the entire function can be pushed down based on its **SHIPPABLE** attribute.
2. If the full function is not pushed down, the optimizer automatically determines which individual SQL statements within the function can be executed separately.

## Function-invoking Statements Without User Tables

When a function is called without referencing the user table, the statement runs on the CN, and the function body is not pushed down. Here is how it works:

1. First, execute the function. Since the statement does not include the user table, it always runs on the CN.
2. Next, execute the statements inside the function body. These can be pushed down if nothing prevents it (using FQS or STREAM plan frameworks).

**CAUTION**

- Calling a function alone (without user tables) with SQL join queries still produces accurate results.
- Pushdown only matters if the function call includes user tables directly.
- Execution plans within the function are not visible by default. Setting **enable\_track\_record\_subsql** to **on** to log them.

**Example: The function body is NOT SHIPPABLE, and the calling statement lacks the user table.**

```
CREATE TABLE student (
 id integer,
 name character varying(50),
 age integer
)
WITH (orientation=row, compression=no)
DISTRIBUTE BY HASH(id);

CREATE TABLE stuscore(
 id integer,
 score integer
)
WITH (orientation=row, compression=no)
DISTRIBUTE BY HASH(id);

INSERT INTO student VALUES (1,'bob',17),(3,'peter',18),(4,'david',15),(8,'alice',17);
INSERT INTO stuscore VALUES (1,88),(3,68),(8,100),(4,90);

--Create a NOT SHIPPABLE function.
CREATE OR REPLACE FUNCTION get_student_id(name character varying)
 RETURNS TABLE(id integer)
 LANGUAGE plpgsql
 VOLATILE NOT FENCED NOT SHIPPABLE
AS $function$
DECLARE
BEGIN
 RETURN QUERY (SELECT id FROM mytest.student WHERE name=$1);
END $function$;
```

Check the execution plan for this function call.

```
EXPLAIN VERBOSE SELECT * FROM get_student_id('alice');
```

Since the user table is missing, the results do not indicate if the function was pushed down, as pushdown only matters when the user table is present.



| QUERY PLAN                                                |                                           |        |            |         |         |
|-----------------------------------------------------------|-------------------------------------------|--------|------------|---------|---------|
| id                                                        | operation                                 | E-rows | E-distinct | E-width | E-costs |
| <hr/>                                                     |                                           |        |            |         |         |
| 1                                                         | -> Function Scan on public.get_student_id | 1000   |            | 4       | 10.25   |
| <hr/>                                                     |                                           |        |            |         |         |
| Predicate Information (identified by plan id)             |                                           |        |            |         |         |
| <hr/>                                                     |                                           |        |            |         |         |
| 1 --Function Scan on public.get_student_id                |                                           |        |            |         |         |
| Function Call: get_student_id('alice'::character varying) |                                           |        |            |         |         |
| <hr/>                                                     |                                           |        |            |         |         |
| Targetlist Information (identified by plan id)            |                                           |        |            |         |         |
| <hr/>                                                     |                                           |        |            |         |         |
| 1 --Function Scan on public.get_student_id                |                                           |        |            |         |         |
| Output: id                                                |                                           |        |            |         |         |

## Function-invoking Statements with User Tables

When a function-calling statement includes a user table, its pushdown capability depends on the function's SHIPPABLE attribute.

**Example: The function attribute is NOT SHIPPABLE and the calling statement contains a user table.**

Check the execution plan to confirm that the function cannot be pushed down.

```
EXPLAIN VERBOSE SELECT get_student_id('alice') FROM stuscore;
```

| QUERY PLAN                                           |                                                   |                                         |  |   |        |
|------------------------------------------------------|---------------------------------------------------|-----------------------------------------|--|---|--------|
| id                                                   | operation                                         | E-rows   E-distinct   E-width   E-costs |  |   |        |
| -----+-----+-----+-----+-----                        |                                                   |                                         |  |   |        |
| 1   ->                                               | Data Node Scan on stuscore "_REMOTE_TABLE_QUERY_" | 30000                                   |  | 0 | 157.35 |
|                                                      |                                                   |                                         |  |   |        |
| SQL Diagnostic Information                           |                                                   |                                         |  |   |        |
| -----                                                |                                                   |                                         |  |   |        |
| SQL is not plan-shipping                             |                                                   |                                         |  |   |        |
| reason: Function get_student_id() can not be shipped |                                                   |                                         |  |   |        |

**Example: The function attribute is SHIPPABLE and the calling statement contains a user table.**

```
--Change the function attribute to SHIPPABLE.
CREATE OR REPLACE FUNCTION get_student_id(name character varying)
 RETURNS TABLE(id integer)
 LANGUAGE plpgsql
 VOLATILE NOT FENCED SHIPPABLE
AS $function$
DECLARE
BEGIN
 RETURN QUERY (SELECT id FROM mytest.student WHERE name=$1);
END $function$;
```

Check the execution plan to confirm that the function can be pushed down.

```
EXPLAIN VERBOSE SELECT get_student_id('alice') FROM stuscore;
```

| QUERY PLAN                                                                                                    |                                          |                                         |  |  |
|---------------------------------------------------------------------------------------------------------------|------------------------------------------|-----------------------------------------|--|--|
| id                                                                                                            | operation                                | E-rows   E-distinct   E-width   E-costs |  |  |
| -----+-----+-----+-----+-----                                                                                 |                                          |                                         |  |  |
| 1   ->                                                                                                        | Data Node Scan on "__REMOTE_FQS_QUERY__" | 0     0   0.00                          |  |  |
|                                                                                                               |                                          |                                         |  |  |
| Targetlist Information (identified by plan id)                                                                |                                          |                                         |  |  |
| -----                                                                                                         |                                          |                                         |  |  |
| 1 -Data Node Scan on "__REMOTE_FQS_QUERY__"                                                                   |                                          |                                         |  |  |
| Output: (get_student_id('alice'::character varying))                                                          |                                          |                                         |  |  |
| Node/s: All datanodes (v3_logical, bucket:32)                                                                 |                                          |                                         |  |  |
| Remote query: SELECT public.get_student_id('alice'::character varying) AS get_student_id FROM public.stuscore |                                          |                                         |  |  |

## Helpful Links

[ALTER FUNCTION](#), [DROP FUNCTION](#)

### 21.13.7.2 ALTER FUNCTION

#### Function

**ALTER FUNCTION** modifies the attributes of a customized function.

#### Precautions

- Only the function owner or a user granted the ALTER permission can run **ALTER FUNCTION**. System administrators have this permission by default.
- To change the function owner, you must be a direct or indirect member of the new owning role.
- To change the function owner, you must have CREATE permission on the function's schemas.
- To change a function's schema, you must also have the CREATE permission on the new schema.
- If a function involves operations on temporary tables, **ALTER FUNCTION** cannot be used.

#### Syntax

- Modify the additional parameter of the custom function.  
**ALTER FUNCTION** function\_name ( ( [ { [ argmode ] [ argname ] argtype } [ , ... ] ] )  
action [ ... ] [ RESTRICT ] ;

The syntax of the **action** clause is as follows:

```
{ CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
| { IMMUTABLE | STABLE | VOLATILE }
```

```
| {SHIPPABLE | NOT SHIPPABLE}
| {NOT FENCED | FENCED}
| {MASKED | NOT MASKED}
| [NOT] LEAKPROOF
| { [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER }
| AUTHID { DEFINER | CURRENT_USER }
| COST execution_cost
| ROWS result_rows
| SET configuration_parameter { { TO | = } { value | DEFAULT } } FROM CURRENT}
| RESET {configuration_parameter | ALL}
```

- Rename a custom function. You can add the schema name before the new function name. However, you cannot change the schema name at the same time.

```
ALTER FUNCTION funname ([{ [argmode] [argname] argtype } [, ...]])
 RENAME TO new_name;
ALTER FUNCTION funname ([{ [argmode] [argname] argtype } [, ...]])
 RENAME TO schema.new_name;
```

- Modify the owner of the custom function.

```
ALTER FUNCTION funname ([{ [argmode] [argname] argtype } [, ...]])
 OWNER TO new_owner;
```

- Modify the schema of the custom function.

```
ALTER FUNCTION funname ([{ [argmode] [argname] argtype } [, ...]])
 SET SCHEMA new_schema;
```

## Parameter Description

Table 21-225 ALTER FUNCTION parameters

| Parameter     | Description                                                                                                         | Value Range                                                                     |
|---------------|---------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| function_name | Specifies the function name to be modified.                                                                         | Name of an existing function.                                                   |
| argmode       | Specifies the mode of an argument. It indicates whether a parameter is an input, output, or input/output parameter. | <b>IN</b> , <b>OUT</b> , or <b>IN OUT</b> .<br>The default value is <b>IN</b> . |
| argname       | Specifies the parameter name.                                                                                       | A string compliant with the <a href="#">identifier naming rules</a> .           |
| argtype       | Specifies the parameter type of a function.                                                                         | Valid types are listed in <a href="#">Data Types</a> .                          |
| new_name      | Specifies the new name of a function.                                                                               | A string compliant with the <a href="#">identifier naming rules</a> .           |
| new_owner     | Specifies the new owner of a function.                                                                              | Valid username or role name.                                                    |
| new_schema    | Specifies the new schema of a function.                                                                             | Existing schema.                                                                |

| Parameter                                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Value Range                                                                                                          |
|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| CALLED ON NULL INPUT   RETURNS NULL ON NULL INPUT   STRICT | <ul style="list-style-type: none"><li>• <b>CALLED ON NULL INPUT</b> declares that some parameters of a function can be invoked in normal mode if the parameter values are <b>NULL</b>. By default, the usage is the same as specifying the parameters.</li><li>• <b>STRICT</b> indicates that the function always returns <b>NULL</b> whenever any of its parameters are <b>NULL</b>. If this parameter is specified, the function is not called when there are <b>NULL</b> parameters; instead a <b>NULL</b> result is returned automatically.</li><li>• The usage of <b>RETURNS NULL ON NULL INPUT</b> is the same as that of <b>STRICT</b>.</li></ul> | -                                                                                                                    |
| IMMUTABLE   STABLE   VOLATILE                              | <ul style="list-style-type: none"><li>• <b>IMMUTABLE</b> indicates that the function always returns the same result if the parameter values are the same.</li><li>• <b>STABLE</b> indicates that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same parameter values, but that its result varies by SQL statements.</li><li>• <b>VOLATILE</b> indicates that the function value can change in one table scanning and no optimization is performed.</li></ul>                                                                                                          | For details about IMMUTABLE, STABLE, and VOLATILE, see <a href="#">Three DWS Function Types Based on Behaviors</a> . |
| SHIPPABLE   NOT SHIPPABLE                                  | <p>Indicates whether the function can be pushed down to DN for execution.</p> <ul style="list-style-type: none"><li>• Functions of the IMMUTABLE type can always be pushed down to the DN.</li><li>• Functions of the STABLE or VOLATILE type can be pushed down to DN only if their attribute is <b>SHIPPABLE</b>.</li></ul>                                                                                                                                                                                                                                                                                                                            | -                                                                                                                    |

| Parameter                            | Description                                                                                                                                                                                                                                             | Value Range                                              |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| MASKED   NOT MASKED                  | Specifies whether the function performs masking when the GUC parameter <b>enable_redactcol_computable</b> is enabled. <b>MASKED</b> supports system function modification and cannot be used together with actions in other <b>ALTER</b> functions.     | -                                                        |
| LEAKPROOF                            | Indicates that the function has no side effect and specifies that the parameter includes only the returned value. <b>LEAKPROOF</b> can be set only by the system administrator.                                                                         | -                                                        |
| EXTERNAL (optional)                  | Used d for SQL compatibility. This function applies to all functions, including external ones.                                                                                                                                                          | -                                                        |
| SECURITY INVOKER/AUTHID CURRENT_USER | Declares that the function will be executed according to the permission of the user that invokes it. By default, the usage is the same as specifying the parameters.<br><b>SECURITY INVOKER</b> and <b>AUTHID CURRENT_USER</b> have the same functions. | -                                                        |
| SECURITY DEFINER/AUTHID DEFINER      | Specifies that the function is to be executed with the permissions of the user that created it. By default, the usage is the same as specifying the parameters.<br>The usage of <b>AUTHID DEFINER</b> is the same as that of <b>SECURITY DEFINER</b> .  | -                                                        |
| COST<br>execution_cost               | A positive number giving the estimated execution cost for the function.<br>The unit of <b>execution_cost</b> is <b>cpu_operator_cost</b> .                                                                                                              | Positive                                                 |
| ROWS result_rows                     | Estimates the number of rows returned by the function. This is only allowed when the function is declared to return a set.                                                                                                                              | A positive number.<br>The default value is <b>1000</b> . |

| Parameter               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Value Range |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| configuration_parameter | <ul style="list-style-type: none"><li>value<br/>Sets a specified database session parameter to a specified value.<ul style="list-style-type: none"><li><b>DEFAULT</b> or <b>RESET</b> indicates that the default settings of the system are used in a new session.</li><li><b>OFF</b> indicates that the setting is disabled.</li></ul></li><li>Specifies the default value.</li><li>from current<br/>Uses the value of <b>configuration_parameter</b> of the current session.</li></ul> | A string    |

## Examples

Create the **func\_add\_sql2** function for calculating the sum of two integers and get the results. If the input is **null**, **null** will be returned.

```
DROP FUNCTION IF EXISTS func_add_sql2;
CREATE FUNCTION func_add_sql2(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/
;
```

Alter the execution rule of the **func\_add\_sql2** function to **IMMUTABLE**. That is, the same result is returned if the parameter is not changed.

```
ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) IMMUTABLE;
```

Change the name of the **func\_add\_sql2** function to **add\_two\_number**.

```
ALTER FUNCTION func_add_sql2(INTEGER, INTEGER) RENAME TO add_two_number;
```

Change the owner of the **add\_two\_number** function to dbadmin.

```
ALTER FUNCTION add_two_number(INTEGER, INTEGER) OWNER TO dbadmin;
```

## Helpful Links

[CREATE FUNCTION](#) and [DROP FUNCTION](#)

### 21.13.7.3 DROP FUNCTION

#### Function

**DROP FUNCTION** deletes an existing function.

## Precautions

- To delete an overloaded function, you must specify the function's parameter (argument) type. For non-overloaded functions, you can delete them by just specifying the function name.
- If a function involves operations on temporary tables, the function cannot be deleted by running **DROP FUNCTION**.

## Syntax

```
DROP FUNCTION [IF EXISTS] function_name
[([{ argmode } [argname] argtype } [, ...]]) [CASCADE | RESTRICT] ;
```

## Parameter Description

**Table 21-226** DROP FUNCTION parameters

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                                                                    |
|--------------------|-----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF EXISTS          | Sends a notice instead of an error if the specified table does not exist.                           | -                                                                                                                                                                                                                                                                              |
| function_name      | Specifies the name of the function to be deleted.                                                   | Use an existing function name.                                                                                                                                                                                                                                                 |
| argmode            | Specifies the mode of a function parameter.                                                         | -                                                                                                                                                                                                                                                                              |
| argname            | Specifies the name of a function parameter.                                                         | -                                                                                                                                                                                                                                                                              |
| argtype            | Specifies the data type of a function parameter.                                                    | -                                                                                                                                                                                                                                                                              |
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: automatically deletes all objects that depend on the function to be deleted (such as operators).</li><li>• <b>RESTRICT</b>: refuses to delete the function if any objects depend on it. This is the default.</li></ul> |

## Examples

Delete the function **add\_two\_number**.

```
DROP FUNCTION add_two_number;
```

## Helpful Links

[ALTER FUNCTION](#), [CREATE FUNCTION](#)



## 21.13.7.4 CREATE PROCEDURE

### Function

**CREATE PROCEDURE** creates a stored procedure.

Stored procedures are sets of precompiled SQL statements that are stored in a database and repeatedly executed.

### Precautions

- When creating a stored procedure, you need to explicitly specify the schema (for example, **schema.table**) that the table object belongs to, or there may be an execution exception.
- **current\_schema** and **search\_path** specified by **SET** during the stored procedure creation are invalid. **search\_path** and **current\_schema** before and after function execution should be the same.
- When creating a stored procedure, it is not allowed to nest other aggregate functions or system functions outside the avg function.
- The precision values of the parameters or return values of a stored procedure are not checked.
- In a cluster with multiple CNs, the input or output parameters of a stored procedure cannot be set to the temporary table type. When a stored procedure is created on a CN that is not currently connected, the correct temporary schema cannot be obtained based on the table name, resulting in an incorrect table type.
- A stored procedure can have multiple return values or no return value. After a stored procedure without a return value is invoked, the command output is empty.
- If a stored procedure has output parameters, the **SELECT** statement uses the default values of the output parameters when calling the procedure. When the **CALL** statement calls the stored procedure, it requires that the output parameter values are adapted to Oracle. When the **CALL** statement calls a non-overloaded function, output parameters must be specified. When the **CALL** statement calls an overloaded PACKAGE function, it can use the default values of the output parameters. For details, see the examples in [CALL](#).
- A stored procedure with the PACKAGE attribute can use overloaded functions.
- Stored procedures cannot be submitted by segment.
- The content in CREATE FUNCTION also applies to CREATE PROCEDURE. For details, see [CREATE FUNCTION](#). After a stored procedure is created, the definition of **CREATE FUNCTION** is returned when the definition is queried.

### Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(([argmode] [argname] argtype [{ DEFAULT | := | = } expression]) [, ...])]
[
 { IMMUTABLE | STABLE | VOLATILE }
 | { SHIPPABLE | NOT SHIPPABLE }
 | { PACKAGE }
 | [NOT] LEAKPROOF
 | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
 | [{ EXTERNAL } SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER | AUTHID DEFINER | AUTHID
```

```
CURRENT_USER}
| COST execution_cost
| ROWS result_rows
| SET configuration_parameter { [TO | =] value | FROM CURRENT }
][...]
{ IS | AS }
plsql_body
/
```

## Parameter Description

**Table 21-227** CREATE PROCEDURE parameters

| Parameter      | Description                                                                                                                                                                                                            | Value Range                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OR REPLACE     | Replaces the original definition when two stored procedures are with the same name.                                                                                                                                    | -                                                                                                                                                                                                                                                                                                                                                                                            |
| procedure_name | Specifies the name of the stored procedure that is created (optionally with schema names).                                                                                                                             | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                                                     |
| argmode        | Specifies the mode of the parameter.<br><b>OUT</b> and <b>INOUT</b> cannot be used in procedure definition of <b>RETURNS TABLE</b> .                                                                                   | <b>IN</b> , <b>OUT</b> , <b>INOUT</b> , or <b>VARIADIC</b> . <ul style="list-style-type: none"><li>• <b>IN</b> (default value): input parameter.</li><li>• <b>OUT</b>: output parameter. Only this type can include <b>VARIADIC</b>.</li><li>• <b>INOUT</b>: both input and output parameters.</li><li>• <b>OUT VARIADIC</b>: output parameter, which is defined as an array type.</li></ul> |
| argname        | Specifies the name of the parameter.<br>There is no strict requirement on the sequence of <b>argname</b> and <b>argmode</b> . You are advised to use <b>argname</b> , <b>argmode</b> , and <b>argtype</b> in sequence. | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                                                     |
| argtype        | Specifies the type of the parameter.                                                                                                                                                                                   | A valid data type.                                                                                                                                                                                                                                                                                                                                                                           |

| Parameter                           | Description                                                                                                                                                 | Value Range |
|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| IMMUTABLE  <br>STABLE  <br>VOLATILE | Specifies a constraint.<br>Parameters here are<br>similar to those of<br><b>CREATE FUNCTION</b> . For<br>details, see <a href="#">CREATE<br/>FUNCTION</a> . | -           |

| Parameter                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Value Range |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| SHIPPABLE   NOT SHIPPABLE | <p>Specifies whether a stored procedure can be pushed down to DNs for execution.</p> <ul style="list-style-type: none"><li>• IMMUTABLE stored procedures can always be pushed down to DNs for execution.</li><li>• STABLE and VOLATILE stored procedures can be pushed down to DNs for execution only when the SHIPPABLE attribute is used.</li></ul> <p>SHIPPABLE indicates that the entire stored procedure will be pushed down to DNs for execution. If SHIPPABLE is incorrectly set, serious problems such as incorrect results may occur.</p> <p>SHIPPABLE, like IMMUTABLE, comes with usage limits. It requires stored procedures to exclude elements that prevent pushdown. Once pushed down to a DN, the stored procedures must rely solely on the DN's dataset in the internal computing.</p> <p>Examples:</p> <ol style="list-style-type: none"><li>1. If a stored procedure references a hash table, you cannot define the procedure as <b>SHIPPABLE</b>.</li><li>2. If factors such as stored procedures and syntax cannot</li></ol> | -           |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Value Range |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
|           | <p>be pushed down, the stored procedure cannot be defined as <b>SHIPPABLE</b>. For details, see Optimizing Statement Pushdown.</p> <p>3. If a stored procedure's calculation process involves data across DNs, the procedure cannot be defined as <b>SHIPPABLE</b>. For example, some aggregation operations involve data across DNs.</p>                                                                                                                                                                                     |             |
| PACKAGE   | <p>Specifies whether the stored procedure can be overloaded. PostgreSQL-style stored procedures can be overloaded, and this parameter is designed for Oracle-style stored procedures.</p> <ul style="list-style-type: none"><li>• All PACKAGE and non-PACKAGE stored procedures cannot be overloaded or replaced.</li><li>• Stored procedures with the PACKAGE attribute functions do not support parameters in the <b>VARIADIC</b> mode.</li><li>• The PACKAGE attribute of a stored procedure cannot be modified.</li></ul> | -           |

| Parameter                              | Description                                                                                                                                                                                                                                                                                                                                                                    | Value Range |
|----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| LEAKPROOF                              | Specifies that the parameters of the stored procedure include only return values.<br><b>LEAKPROOF</b> can be set only by the system administrator.                                                                                                                                                                                                                             | -           |
| CALLED ON NULL INPUT                   | Declares that some parameters of the stored procedure can be called in normal mode if the parameter values are <b>NULL</b> . It can be omitted.                                                                                                                                                                                                                                | -           |
| RETURNS NULL ON NULL INPUT   STRICT    | <b>STRICT</b> specifies that the stored procedure always returns <b>NULL</b> whenever any of its parameters are <b>NULL</b> . If it is specified, the stored procedure is not executed when there are <b>NULL</b> parameters; instead a <b>NULL</b> result is returned automatically.<br>The usage of <b>RETURNS NULL ON NULL INPUT</b> is the same as that of <b>STRICT</b> . | -           |
| EXTERNAL                               | The keyword <b>EXTERNAL</b> is allowed for SQL conformance, but it is optional since, unlike in SQL, this feature applies to all stored procedures not only external ones.                                                                                                                                                                                                     | -           |
| SECURITY INVOKER   AUTHID CURRENT_USER | Specifies that the stored procedure is to be executed with the permissions of the user that calls it. It can be omitted.<br><b>SECURITY INVOKER</b> and <b>AUTHID CURRENT_USER</b> have the same functions.                                                                                                                                                                    | -           |

| Parameter                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range                                                        |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| SECURITY DEFINER   AUTHID DEFINER | Specifies that the stored procedure will be executed with the permissions of the user who created it.<br><b>AUTHID DEFINER</b> and <b>SECURITY DEFINER</b> have the same functions.                                                                                                                                                                                                                                                                                                                                                                                                               | -                                                                  |
| COST<br>execution_cost            | Estimates the execution cost of the stored procedure. The unit of <b>execution_cost</b> is <b>cpu_operator_cost</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Positive                                                           |
| ROWS result_rows                  | Estimates the number of rows returned by the stored procedure. This is only allowed when the stored procedure is declared to return a set.                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Its value is a positive number. The default value is <b>1000</b> . |
| configuration_parameter           | Specifies parameter values configured in the stored procedure. <ul style="list-style-type: none"><li>• <b>value</b><br/>Sets a specified database session parameter to a specified value. Value range: a string<br/>Specifies the default value.<ul style="list-style-type: none"><li>– <b>DEFAULT</b>: The setting will be used in new sessions.</li><li>– <b>OFF</b>: The setting is disabled.</li><li>– <b>RESET</b>: The setting will be used in new sessions.</li></ul></li><li>• <b>from current</b><br/>Uses the value of <b>configuration_parameter</b> of the current session.</li></ul> | -                                                                  |

| Parameter  | Description                                                                                                                                                                                                                                                                                      | Value Range |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| plsql_body | Specifies the PL/SQL stored procedure body.<br>When you create a user, or perform other operations requiring password input in a stored procedure, the system catalog and csv log records the unencrypted password. You are advised not to perform such operations in the stored procedure body. | -           |

## Examples

- Create the stored procedure **prc\_add**, where **param1** is an input parameter and **param2** is an input and output parameter. The procedure calculates the sum of **param1** and **param2**, assigns the result to **param2**, and outputs the result.

```
CREATE OR REPLACE PROCEDURE prc_add
(
 param1 IN INTEGER,
 param2 IN OUT INTEGER
)
AS
BEGIN
 param2:= param1 + param2;
 dbms_output.put_line('result is: '||to_char(param2));
END;
/
```

Call the stored procedure **prc\_add**.

```
CALL prc_add(2,3);
```

- Create the stored procedure **pro\_variadic**. The parameter mode of **param1** is **VARIADIC**. The integer array is converted to a TEXT format and then output via **param2**.

```
CREATE OR REPLACE PROCEDURE pro_variadic (param1 VARIADIC int4[],param2 OUT TEXT)
AS
BEGIN
 param2:= param1::text;
END;
/
```

Execute the stored procedure **pro\_variadic**.

```
SELECT pro_variadic(VARIADIC param1=> array[1,2,3,4]);
```

- Create the stored procedure **package\_func\_overload** with the PACKAGE attribute.

```
CREATE OR REPLACE PROCEDURE package_func_overload(col int, col2 out varchar)
PACKAGE
AS
DECLARE
 col_type text;
BEGIN
 col2 := '122';
```



```
dbms_output.put_line('two varchar parameters ' || col2);
END;
/
```

## Helpful Links

For details about how to delete a stored procedure, see [DROP PROCEDURE](#). For details about how to call a stored procedure, see [CALL](#).

### 21.13.7.5 DROP PROCEDURE

#### Function

**DROP PROCEDURE** deletes an existing stored procedure.

#### Precautions

None.

#### Syntax

```
DROP PROCEDURE [IF EXISTS] procedure_name ;
```

#### Parameter Description

**Table 21-228** DROP PROCEDURE parameters

| Parameter      | Description                                                                | Value Range                             |
|----------------|----------------------------------------------------------------------------|-----------------------------------------|
| IF EXISTS      | Sends a notice instead of an error if the stored procedure does not exist. | -                                       |
| procedure_name | Specifies the name of the stored procedure to be deleted.                  | An existing name of a stored procedure. |

#### Examples

Delete a stored procedure.

```
DROP PROCEDURE prc_add;
```

## Helpful Links

[CREATE PROCEDURE](#)

### 21.13.7.6 CREATE TRIGGER

#### Function

Creates a trigger. The trigger will be associated with a specified table or view, and will execute a specified function when certain events occur.

## Precautions

- Currently, triggers can be created only on ordinary row-store tables, instead of on column-store tables, temporary tables, or unlogged tables.
- If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name.
- A trigger works only on one table. There is no limit on the number of triggers that can be created. However, more triggers on a table consume more performance.
- Triggers are usually used for data association and synchronization between multiple tables. SQL execution performance is greatly affected. Therefore, you are advised not to use this statement when a large amount of data needs to be synchronized and performance requirements are high.
- When a trigger meets the following conditions, the trigger statement and trigger itself can be pushed together down to a DN for execution, improving the trigger execution performance:
  - **enable\_trigger\_shipping** and **enable\_fast\_query\_shipping** are both enabled. (This is the default configuration.)
  - The trigger function used by the source table is a PL/pgSQL function (recommended).
  - The source and target tables have the same type and number of distribution keys, are both row-store tables, and belong to the same Node Group.
  - The **INSERT**, **UPDATE**, or **DELETE** statement on the source table contains an expression about equality comparison between all the distribution keys and the *NEW* or *OLD* variable.
  - The **INSERT**, **UPDATE**, or **DELETE** statement on the source table can be pushed down without a trigger.
  - There are only six types of triggers, specified by **INSERT/UPDATE/DELETE**, **AFTER/BEFORE**, and **FOR EACH ROW**, on the source table, and all the triggers can be pushed down.
- The following details trigger types:
  - **INSTEAD OF** triggers must be marked as **FOR EACH ROW** and can be defined only on views.
  - **BEFORE** and **AFTER** triggers on a view must be marked as **FOR EACH STATEMENT**.
  - **TRUNCATE** triggers must be marked as **FOR EACH STATEMENT**.

## Syntax

```
CREATE [CONSTRAINT] TRIGGER trigger_name { BEFORE | AFTER | INSTEAD OF } { event [OR ...] }
ON table_name
[FROM referenced_table_name]
{ NOT DEFERRABLE | [DEFERRABLE] } { INITIALLY IMMEDIATE | INITIALLY DEFERRED } }
[FOR [EACH] { ROW | STATEMENT }]
[WHEN (condition)]
EXECUTE PROCEDURE function_name (arguments);
```

Events include:

```
INSERT
UPDATE [OF column_name [, ...]]
```

DELETE  
TRUNCATE

## Parameter Description

**Table 21-229** CREATE TRIGGER parameters

| Parameter    | Description                                                                                                                                                                                                                                                                                                                                      | Value Range                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| CONSTRAINT   | Creates a constraint trigger, that is, a trigger is used as a constraint. Such a trigger is similar to a regular trigger except that the timing of the trigger firing can be adjusted using <b>SET CONSTRAINTS</b> . Constraint triggers must be <b>AFTER ROW</b> triggers.                                                                      | -                                                                        |
| trigger_name | Specifies the name of the trigger to be created.<br><br>The name cannot be schema-qualified because the trigger inherits the schema of its table. In addition, triggers on the same table cannot be named the same. For a constraint trigger, this is also the name to use when you modify the trigger's behavior using <b>SET CONSTRAINTS</b> . | A string, which must comply with the <a href="#">naming convention</a> . |
| BEFORE       | Specifies that a trigger function is called before the trigger event.                                                                                                                                                                                                                                                                            | -                                                                        |
| AFTER        | Specifies that a trigger function is called after the trigger event. A constraint trigger can only be specified as <b>AFTER</b> .                                                                                                                                                                                                                | -                                                                        |
| INSTEAD OF   | Specifies that a trigger function directly replaces the trigger event.                                                                                                                                                                                                                                                                           | -                                                                        |

| Parameter                                | Description                                                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| event                                    | Event for starting a trigger.                                                                                                                                                                                     | <p>Values are <b>INSERT</b>, <b>UPDATE</b>, <b>DELETE</b>, and <b>TRUNCATE</b>. You can also specify multiple trigger events through <b>OR</b>.</p> <p>For <b>UPDATE</b> events, use the following syntax to specify a list of columns:<br/>UPDATE OF column_name1<br/>[, column_name2 ... ]</p> <p>The trigger will only fire if at least one of the listed columns is mentioned as a target of the <b>UPDATE</b> statement. <b>INSTEAD OF UPDATE</b> events do not support lists of columns.</p> |
| table_name                               | Specifies the name of the table where a trigger needs to be created.                                                                                                                                              | Name of an existing table in the database.                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| referenced_table_name                    | Specifies the name of another table referenced by a constraint. This parameter can be specified only for constraint triggers. It does not support foreign key constraints and is not recommended for general use. | Name of an existing table in the database.                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| DEFERRABLE   NOT DEFERRABLE              | Specifies the start time of the trigger. It can only be specified for constraint triggers. Controls whether the constraint can be deferred.<br>For details, see <a href="#">CREATE TABLE</a> .                    | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| INITIALLY IMMEDIATE   INITIALLY DEFERRED | If a constraint is deferrable, the two clauses specify the default time to check the constraint, and can be specified only for constraint triggers.<br>For details, see <a href="#">CREATE TABLE</a> .            | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

| Parameter                         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Value Range |
|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| FOR EACH ROW   FOR EACH STATEMENT | <p>Specifies the frequency of firing a trigger.</p> <ul style="list-style-type: none"><li>• <b>FOR EACH ROW</b> indicates that the trigger should be fired once for every row affected by the trigger event.</li><li>• <b>FOR EACH STATEMENT</b> indicates that the trigger should be fired just once per SQL statement.</li></ul> <p>If this parameter is not specified, the default value <b>FOR EACH STATEMENT</b> will be used. Constraint triggers can only be specified as <b>FOR EACH ROW</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | -           |
| condition                         | <p>Specifies a Boolean expression that determines whether a trigger function will actually be executed. If <b>WHEN</b> is specified, the function will be called only when <b>condition</b> returns <b>true</b>.</p> <ul style="list-style-type: none"><li>• In <b>FOR EACH ROW</b> triggers, the <b>WHEN</b> condition can reference the columns of old or new row values by writing <b>OLD.column_name</b> or <b>NEW.column_name</b>, respectively. Note that <b>INSERT</b> triggers cannot reference <b>OLD</b> and <b>DELETE</b> triggers cannot reference <b>NEW</b>.</li><li>• <b>INSTEAD OF</b> triggers do not support <b>WHEN</b> conditions.</li><li>• <b>WHEN</b> expressions cannot contain subqueries.</li><li>• For constraint triggers, evaluation of the <b>WHEN</b> condition is not deferred, but occurs immediately after the update operation is performed. If the condition does not return <b>true</b>, the trigger will not be queued for deferred execution.</li></ul> | -           |
| function_name                     | <p>Specifies a user-defined function, which must be declared as taking no parameters and returning data of the trigger type. This function is executed when a trigger fires.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | -           |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                | Value Range |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| arguments | Specifies an optional, comma-separated list of parameters to be provided to a function when a trigger is executed. Parameters are literal string constants. Simple names and numeric constants can also be included, but they will all be converted to strings. Check descriptions of the implementation language of a trigger function to find out how these parameters are accessed within the function. | -           |

**Table 21-230** Types of triggers supported on tables and views

| Trigger Timing | 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           |
| INSTEAD OF     | INSERT/UPDATE/DELETE | Views         | Not supported    |
|                | TRUNCATE             | Not supported | Not supported    |

**Table 21-231** Special variables in the functions PL/pgSQL triggers

| Variable | Description                                                                                               |
|----------|-----------------------------------------------------------------------------------------------------------|
| NEW      | New tuple for <b>INSERT/UPDATE</b> operations. This variable is <b>NULL</b> for <b>DELETE</b> operations. |
| OLD      | Old tuple for <b>UPDATE/DELETE</b> operations. This variable is <b>NULL</b> for <b>INSERT</b> operations. |
| TG_NAME  | Trigger name                                                                                              |
| TG_WHEN  | Trigger timing ( <b>BEFORE/AFTER/INSTEAD OF</b> )                                                         |

| Variable        | Description                                                                                                              |
|-----------------|--------------------------------------------------------------------------------------------------------------------------|
| TG_LEVEL        | Trigger frequency ( <b>ROW/STATEMENT</b> )                                                                               |
| TG_OP           | Trigger event ( <b>INSERT/UPDATE/DELETE/TRUNCATE</b> )                                                                   |
| TG_RELID        | OID of the table where a trigger is located                                                                              |
| TG_RELNAME      | Name of the table where a trigger is located. (This variable is now discarded and is replaced by <b>TG_TABLE_NAME</b> .) |
| TG_TABLE_NAME   | Name of the table where a trigger is located.                                                                            |
| TG_TABLE_SCHEMA | Schema information of the table where a trigger is located                                                               |
| TG_NARGS        | Number of parameters for a trigger function                                                                              |
| TG_ARGV[]       | List of parameters for a trigger function                                                                                |

## Examples

Create a source table and a trigger table:

```
CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);
```

Create the trigger function **tri\_insert\_func()**:

```
CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
 INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
 RETURN NEW;
END
$$ LANGUAGE PLPGSQL;
```

Create the trigger function **tri\_update\_func()**:

```
CREATE OR REPLACE FUNCTION tri_update_func() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
 UPDATE test_trigger_des_tbl SET id3 = NEW.id3 WHERE id1=OLD.id1;
 RETURN OLD;
END
$$ LANGUAGE PLPGSQL;
```

Create the trigger function **tri\_delete\_func()**:

```
CREATE OR REPLACE FUNCTION tri_delete_func() RETURNS TRIGGER AS
$$
DECLARE
BEGIN
```

```
DELETE FROM test_trigger_des_tbl WHERE id1=OLD.id1;
RETURN OLD;
END
$$ LANGUAGE PLPGSQL;
```

Create an **INSERT** trigger:

```
CREATE TRIGGER insert_trigger
BEFORE INSERT ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_insert_func();
```

Create an **UPDATE** trigger:

```
CREATE TRIGGER update_trigger
AFTER UPDATE ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_update_func();
```

Create a **DELETE** trigger:

```
CREATE TRIGGER delete_trigger
BEFORE DELETE ON test_trigger_src_tbl
FOR EACH ROW
EXECUTE PROCEDURE tri_delete_func();
```

## Helpful Links

[ALTER TRIGGER](#), [DROP TRIGGER](#), [ALTER TABLE](#)

### 21.13.7.7 ALTER TRIGGER

#### Function

**ALTER TRIGGER** modifies the definition of a trigger.

#### Precautions

Only the owner of a table where a trigger is created and system administrators can run the **ALTER TRIGGER** statement.

#### Syntax

```
ALTER TRIGGER trigger_name ON table_name RENAME TO new_name;
```

#### Parameter Description

**Table 21-232** ALTER TRIGGER parameters

| Parameter    | Description                                                                  | Value Range                                           |
|--------------|------------------------------------------------------------------------------|-------------------------------------------------------|
| trigger_name | Specifies the name of the trigger to be modified.                            | Valid trigger name.                                   |
| table_name   | Specifies the name of the table where the trigger to be modified is located. | Name of the valid table where the trigger is located. |



| Parameter | Description                     | Value Range                                                           |
|-----------|---------------------------------|-----------------------------------------------------------------------|
| new_name  | Specifies the new trigger name. | A string compliant with the <a href="#">identifier naming rules</a> . |

## Examples

Create sample tables **test\_trigger\_src\_tbl** and **test\_trigger\_des\_tbl**.

```
DROP TABLE IF EXISTS test_trigger_src_tbl;
DROP TABLE IF EXISTS test_trigger_des_tbl;

CREATE TABLE test_trigger_src_tbl(id1 INT, id2 INT, id3 INT);
CREATE TABLE test_trigger_des_tbl(id1 INT, id2 INT, id3 INT);
```

Create the **tri\_insert\_func()** trigger function.

```
DROP FUNCTION IF EXISTS tri_insert_func;
CREATE OR REPLACE FUNCTION tri_insert_func() RETURNS TRIGGER AS
$$
 DECLARE
 BEGIN
 INSERT INTO test_trigger_des_tbl VALUES(NEW.id1, NEW.id2, NEW.id3);
 RETURN NEW;
 END
$$ LANGUAGE PLPGSQL;
```

Create the **insert\_trigger** trigger.

```
CREATE TRIGGER insert_trigger
 BEFORE INSERT ON test_trigger_src_tbl
 FOR EACH ROW
 EXECUTE PROCEDURE tri_insert_func();
```

Modify the **insert\_trigger** trigger.

```
ALTER TRIGGER insert_trigger ON test_trigger_src_tbl RENAME TO delete_trigger_renamed;
```

Disable the **delete\_trigger\_renamed** trigger.

```
ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER delete_trigger_renamed;
```

Disable all triggers on the **test\_trigger\_src\_tbl** table.

```
ALTER TABLE test_trigger_src_tbl DISABLE TRIGGER ALL;
```

## Helpful Links

[CREATE TRIGGER](#), [DROP TRIGGER](#), and [ALTER TABLE](#)

### 21.13.7.8 DROP TRIGGER

#### Function

**DROP TRIGGER** deletes a trigger.

## Precautions

Only the owner of a trigger and system administrators can run the **DROP TRIGGER** statement.

## Syntax

```
DROP TRIGGER [IF EXISTS] trigger_name ON table_name [CASCADE | RESTRICT];
```

## Parameter Description

**Table 21-233** DROP TRIGGER parameters

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                              |
|--------------------|-----------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IF EXISTS          | Sends a notice instead of an error if the specified trigger does not exist.                         | -                                                                                                                                                                                                                        |
| trigger_name       | Specifies the name of the trigger to be deleted.                                                    | An existing trigger.                                                                                                                                                                                                     |
| table_name         | Specifies the name of the table where the trigger to be deleted is located.                         | An existing table having a trigger                                                                                                                                                                                       |
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: Deletes objects that depend on the trigger.</li><li>• <b>RESTRICT</b>: Refuses to delete the trigger if any objects depend on it. This is the default.</li></ul> |

## Examples

Delete trigger **insert\_trigger**.

```
DROP TRIGGER insert_trigger ON test_trigger_src_tbl;
```

## Helpful Links

[CREATE TRIGGER](#), [ALTER TRIGGER](#), [ALTER TABLE](#)

### 21.13.7.9 FETCH

## Function

**FETCH** retrieves data using a previously-created cursor.

A cursor has an associated position, which is used by **FETCH**. The cursor position can be before the first row of the query result, on any particular row of the result, or after the last row of the result.

- When created, a cursor is positioned before the first row.
- After fetching some rows, the cursor is positioned on the row most recently retrieved.
- If **FETCH** runs off the end of the available rows then the cursor is left positioned after the last row, or before the first row if fetching backward.
- **FETCH ALL** or **FETCH BACKWARD ALL** will always leave the cursor positioned after the last row or before the first row.

## Precautions

- If **NO SCROLL** is defined for the cursor, a backward fetch like **FETCH BACKWARD** is not allowed.
- The forms **NEXT**, **PRIOR**, **FIRST**, **LAST**, **ABSOLUTE**, and **RELATIVE** appropriately fetch a record after moving the cursor. If the cursor is already after the last row before being moved, an empty result is returned, and the cursor is left positioned before the first row (backward fetch) or after the last row (forward fetch) as appropriate.
- The forms using **FORWARD** and **BACKWARD** retrieve the indicated number of rows moving in the forward or backward direction, leaving the cursor positioned on the last-returned row (or after (backward fetch)/before (forward fetch) all rows, if the count exceeds the number of rows available).
- **RELATIVE 0**, **FORWARD 0**, and **BACKWARD 0** all request fetching the current row without moving the cursor, that is, re-fetching the most recently fetched row. This will succeed unless the cursor is positioned before the first row or after the last row, in which case, no row is returned.
- If the cursor of **FETCH** involves a column-store table, backward fetches like **BACKWARD**, **PRIOR**, and **FIRST** are not supported.

## Syntax

```
FETCH [direction { FROM | IN }] cursor_name;
```

The **direction** clause specifies optional parameters.

```
NEXT
| PRIOR
| FIRST
| LAST
| ABSOLUTE count
| RELATIVE count
| count
| ALL
| FORWARD
| FORWARD count
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```

## Parameter Description

**Table 21-234** FETCH parameters

| Parameter                    | Description                                                            | Value Range                                     |
|------------------------------|------------------------------------------------------------------------|-------------------------------------------------|
| direction_clause             | Defines the fetch direction.                                           | For details, see <a href="#">Table 21-235</a> . |
| { FROM   IN }<br>cursor_name | Specifies the cursor name using the keyword <b>FROM</b> or <b>IN</b> . | Name of the cursor                              |

**Table 21-235** Value range of direction\_clause

| Value Range                    | Description                                                                                                                                                                                                                            | Remarks                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>NEXT</b><br>(default value) | Fetches the next row.                                                                                                                                                                                                                  | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>PRIOR</b>                   | Fetches the prior row.                                                                                                                                                                                                                 | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>FIRST</b>                   | Fetches the first row of the query (same as <b>ABSOLUTE 1</b> ).                                                                                                                                                                       | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>LAST</b>                    | Fetches the last row of the query (same as <b>ABSOLUTE -1</b> ).                                                                                                                                                                       | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <b>ABSOLUTE</b><br>count       | Fetches the ( <b>count</b> )'th row of the query.<br><b>ABSOLUTE</b> fetches are not any faster than navigating to the desired row with a relative move: the underlying implementation must traverse all the intermediate rows anyway. | <b>count</b> is a possibly-signed integer constant: <ul style="list-style-type: none"><li>• If <b>count</b> is a positive integer, fetches the (<b>count</b>)'th row of the query, starting from the first row. If <b>count</b> is less than the current cursor position, a <b>rewind</b> operation is required, which is currently not supported.</li><li>• If <b>count</b> is a negative value or 0, a backward scanning is required, which is currently not supported.</li></ul> |

| Value Range    | Description                                                                                 | Remarks                                                                                                                                                                                                                                                                                                                                                    |
|----------------|---------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RELATIVE count | Fetches the (count)'th succeeding row, or the abs(count)'th prior row if count is negative. | <b>count</b> is a possibly-signed integer constant: <ul style="list-style-type: none"><li>• If <b>count</b> is a positive integer, fetches the (count)'th succeeding row.</li><li>• If <b>count</b> is a negative value, a backward scanning is required, which is currently not supported.</li><li>• <b>RELATIVE 0</b> fetches the current row.</li></ul> |
| count          | Fetches the next <b>count</b> rows (same as <b>FORWARD count</b> ).                         | -                                                                                                                                                                                                                                                                                                                                                          |
| ALL            | Fetches all remaining rows (same as <b>FORWARD ALL</b> ).                                   | -                                                                                                                                                                                                                                                                                                                                                          |
| FORWARD        | Fetches the next row (same as <b>NEXT</b> ).                                                | -                                                                                                                                                                                                                                                                                                                                                          |
| FORWARD count  | Fetches the next or prior <b>count</b> rows (same as <b>RELATIVE count</b> ).               | -                                                                                                                                                                                                                                                                                                                                                          |
| FORWARD ALL    | Fetches all remaining rows.                                                                 | -                                                                                                                                                                                                                                                                                                                                                          |
| BACKWARD       | Fetches the prior row (same as <b>PRIOR</b> ).                                              | -                                                                                                                                                                                                                                                                                                                                                          |
| BACKWARD count | Fetches the prior <b>count</b> rows (scanning backwards).                                   | <b>count</b> is a possibly-signed integer constant: <ul style="list-style-type: none"><li>• If <b>count</b> is a positive integer, fetches the (count)'th prior row.</li><li>• If <b>count</b> is a negative integer, fetches the abs(count)'th succeeding row.</li><li>• <b>BACKWARD 0</b> re-fetches the current row.</li></ul>                          |
| BACKWARD ALL   | Fetches all prior rows (scanning backwards).                                                | -                                                                                                                                                                                                                                                                                                                                                          |

## Examples

Example 1: Run the **SELECT** statement to read a table using a cursor.

Set up the **cursor1** cursor.

```
CURSOR cursor1 FOR SELECT * FROM tpods.customer_address ORDER BY 1;
```

Fetch the first three rows from **cursor1**.

```
FETCH FORWARD 3 FROM cursor1;
```

Example 2: Use a cursor to read the content in the **VALUES** clause.

Set up the cursor **cursor2**.

```
CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;
```

Fetch the first two rows from **cursor2**.

```
FETCH FORWARD 2 FROM cursor2;
```

## Helpful Links

[CLOSE](#), [MOVE](#)

### 21.13.7.10 MOVE

## Function

**MOVE** repositions a cursor without retrieving any data. **MOVE** works exactly like the **FETCH** command, except it only repositions the cursor and does not return rows.

## Precautions

None

## Syntax

```
MOVE [direction [FROM | IN]] cursor_name;
```

The **direction** clause specifies optional parameters.

```
NEXT
| PRIOR
| FIRST
| LAST
| ABSOLUTE count
| RELATIVE count
| count
| ALL
| FORWARD
| FORWARD count
| FORWARD ALL
| BACKWARD
| BACKWARD count
| BACKWARD ALL
```

## Parameter Description

**MOVE** command parameters are the same as **FETCH** command parameters. For details, see [Parameter Description](#) in **FETCH**.

 **NOTE**

On successful completion, a **MOVE** command returns a command tag of the form **MOVE count**. The **count** is the number of rows that a **FETCH** command with the same parameters would have returned (possibly zero).

## Examples

Create table **reason** and insert data into it.

```
DROP TABLE IF EXISTS reason;
CREATE TABLE reason
(
 a int primary key,
 b int,
 c int
);

INSERT INTO reason VALUES (1, 2, 3);
```

Start a transaction.

```
START TRANSACTION;
```

Define **cursor1** cursor.

```
CURSOR cursor1 FOR SELECT * FROM tpceds.reason;
```

Skip the first three rows of **cursor1**.

```
MOVE FORWARD 3 FROM cursor1;
```

Fetch the first four rows from **cursor1**.

```
FETCH 4 FROM cursor1;
```

Close a cursor.

```
CLOSE cursor1;
```

End the transaction.

```
END;
```

## Helpful Links

[CLOSE](#), [FETCH](#)

### 21.13.7.11 CURSOR

#### Function

This statement is used to create a cursor and retrieve specified rows of data from a query.

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

#### Precautions

- **CURSOR** is used only in transaction blocks.

- Generally, **CURSOR** and **SELECT** both have text returns. Since data is stored in binary format in the system, the system needs to convert the data from the binary format to the text format. If data is returned in text format, the client-end application needs to convert the data back to a binary format for processing. **FETCH** implements conversion between binary data and text data.
- Use a binary cursor unless necessary, since a text cursor occupies larger storage space than a binary cursor. A binary cursor returns internal binary data, which is easier to operate. To return data in text format, it is advisable to retrieve data in text format, therefore reducing workload at the client end. For example, the value 1 in an integer column of a query is returned as a character string 1 if a default cursor is used, but is returned as a 4-byte binary value (big-endian) if a binary cursor is used.

## Syntax

```
CURSOR cursor_name
[BINARY] [NO SCROLL] [{ WITH | WITHOUT } HOLD]
FOR query ;
```

## Parameter Description

Table 21-236 CURSOR parameters

| Parameter                | Description                                                                                                                                                                                                                                                                                                                             | Value Range                                                              |
|--------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| cursor_name              | Specifies the name of the cursor to be created.                                                                                                                                                                                                                                                                                         | A string, which must comply with the <a href="#">naming convention</a> . |
| BINARY                   | Specifies that data retrieved by the cursor will be returned in binary format, not in text format.                                                                                                                                                                                                                                      | -                                                                        |
| NO SCROLL                | Specifies the mode of data retrieval by the cursor. <ul style="list-style-type: none"><li>• <b>NO SCROLL</b>: If <b>NO SCROLL</b> is specified, backward fetches will be rejected.</li><li>• Not stated: The system automatically determines whether the cursor can be used for backward fetches based on the execution plan.</li></ul> | -                                                                        |
| WITH HOLD   WITHOUT HOLD | Specifies whether the cursor can still be used after the cursor creation event. <ul style="list-style-type: none"><li>• <b>WITH HOLD</b> indicates that the cursor can still be used.</li><li>• <b>WITHOUT HOLD</b> indicates that the cursor cannot be used. (default setting)</li></ul>                                               | -                                                                        |



| Parameter | Description                                                                             | Value Range                                    |
|-----------|-----------------------------------------------------------------------------------------|------------------------------------------------|
| query     | The <b>SELECT</b> or <b>VALUES</b> clause specifies the row to return the cursor value. | Use the <b>SELECT</b> or <b>VALUES</b> clause. |

## Examples

Start a transaction.

```
START TRANSACTION;
```

Create a cursor named **cursor1**:

```
CURSOR cursor1 FOR SELECT * FROM tpods.customer_address ORDER BY 1;
```

create a cursor named **cursor2**:

```
CURSOR cursor2 FOR VALUES(1,2),(0,3) ORDER BY 1;
```

Usage of the **WITH HOLD** cursor:

1. Set up the **WITH HOLD** cursor.  

```
DECLARE cursor3 CURSOR WITH HOLD FOR SELECT * FROM tpods.customer_address ORDER BY 1;
```
2. Fetch the first two rows from cursor3.  

```
FETCH FORWARD 2 FROM cursor3;
```
3. End the transaction.  

```
END;
```
4. Fetch the next row from cursor3.  

```
FETCH FORWARD 1 FROM cursor3;
```
5. Close the cursor.  

```
CLOSE cursor3;
```

## Helpful Links

[FETCH](#)

### 21.13.7.12 CLOSE

#### Function

**CLOSE** frees the resources associated with a cursor.

#### Precautions

- After a cursor is closed, no subsequent operations are allowed on it.
- Cursors that are no longer used should be closed.
- Each non-held open cursor should be implicitly closed after the transaction that created the cursor is terminated with **COMMIT** or **ROLLBACK**.
- A holdable cursor is implicitly closed when the transaction that creates the cursor exits through **ROLLBACK**.
- If the transaction that creates a cursor is committed, the cursor remains open until an explicit **CLOSE** operation is performed or the client is disconnected.

- DWS does not have an explicit **OPEN** cursor statement. A cursor opens when you define it with the **CURSOR** command. You can see all available cursors by querying the **pg\_cursors** system view.

## Syntax

```
CLOSE { cursor_name | ALL };
```

## Parameter Description

Table 21-237 CLOSE parameters

| Parameter   | Description                                    | Value Range                 |
|-------------|------------------------------------------------|-----------------------------|
| cursor_name | Specifies the name of the cursor to be closed. | Name of an existing cursor. |
| ALL         | Closes all open cursors.                       | -                           |

## Examples

Close a cursor.

```
CLOSE cursor1;
```

## Links

[FETCH](#) and [MOVE](#)

## 21.13.8 Data Masking and Security

### 21.13.8.1 CREATE ROW LEVEL SECURITY POLICY

#### Function

**CREATE ROW LEVEL SECURITY POLICY** creates a row-level access control policy for a table.

The policy takes effect only after row-level access control is enabled (by running **ALTER TABLE... ENABLE ROW LEVEL SECURITY** | **ALTER FOREIGN TABLE ... ENABLE ROW LEVEL SECURITY**).

Currently, row-level access control affects the read (**SELECT**, **UPDATE**, **DELETE**) of data tables and does not affect the write (**INSERT** and **MERGE INTO**) of data tables. The table owner or system administrators can create an expression in the **USING** clause. When the client reads the data table, the database server combines the expressions that meet the condition and applies it to the execution plan in the statement rewriting phase of a query. For each tuple in a data table, if the expression returns **TRUE**, the tuple is visible to the current user; if the expression returns **FALSE** or **NULL**, the tuple is invisible to the current user.

A row-level access control policy name is specific to a table. A data table cannot have row-level access control policies with the same name. Different data tables can have the same row-level access control policy.

Row-level access control policies can be applied to specified operations (**SELECT**, **UPDATE**, **DELETE**, and **ALL**). **ALL** indicates that **SELECT**, **UPDATE**, and **DELETE** will be affected. For a new row-level access control policy, the default value **ALL** will be used if you do not specify the operations that will be affected.

Row-level access control policies can be applied to a specified user (role) or to all users (**PUBLIC**). For a new row-level access control policy, the default value **PUBLIC** will be used if you do not specify the user that will be affected.

## Precautions

- Row-level access control policies can be defined for row-store tables, row-store partitioned tables, column-store tables, column-store partitioned tables, replication tables, unlogged tables, hash tables, and non-EXTERNAL SCHEMA foreign tables.
- Row-level access control policies cannot be defined for HDFS tables, EXTERNAL SCHEMA foreign tables, and temporary tables.
- Row-level access control policies cannot be defined for views.
- A maximum of 100 row-level access control policies cannot be defined for a table.
- Users with administrator permissions, initial O&M users (Ruby), table owners, and members of the table owner role are not affected by row-level access control and can view full data of the table.
- Tables queried by using SQL statements, views, functions, and stored procedures are affected by row-level access control policies.
- The type of a column on which a row-level access control policy depends cannot be changed. For example, the following modifications are not supported:

```
ALTER TABLE public.all_data ALTER COLUMN role TYPE text;
```

## Syntax

```
CREATE [ROW LEVEL SECURITY] POLICY policy_name ON table_name
[AS { PERMISSIVE | RESTRICTIVE }]
[FOR { ALL | SELECT | UPDATE | DELETE }]
[TO { role_name | PUBLIC } [, ...]]
USING (using_expression)
```

## Parameter Description

**Table 21-238** CREATE ROW LEVEL SECURITY POLICY parameters

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| policy_name | Specifies the name of a row-level access control policy to be created. The names of row-level access control policies for a table must be unique.                                                                                                                                                                                                                                                                                                                                                                 | -           |
| table_name  | Specifies the name of a table to which a row-level access control policy is applied.                                                                                                                                                                                                                                                                                                                                                                                                                              | -           |
| PERMISSIVE  | Specifies that the row-level access control policy is to be created as a permissive policy. For a given query, all applicable permissive policies are combined using the OR operator. Row-level access control policies are permissive by default.                                                                                                                                                                                                                                                                | -           |
| RESTRICTIVE | Specifies that the row-level access control policy is to be created as a restrictive policy. For a given query, all applicable restrictive policies are combined using the AND operator.<br><br>At least one permissive policy is required to grant access to data records. If only restrictive policies are used, no records will be accessible. When both permissive and restrictive policies are used, a record is accessible only when it passes at least one permissive policy and all restrictive policies. | -           |

| Parameter | Description                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| command   | Specifies the SQL operations affected by a row-level access control policy, including <b>ALL</b> , <b>SELECT</b> , <b>UPDATE</b> , and <b>DELETE</b> . If this parameter is not specified, the default value <b>ALL</b> will be used, covering <b>SELECT</b> , <b>UPDATE</b> , and <b>DELETE</b> . | <p>If <i>command</i> is set to <b>SELECT</b>, only tuple data that meets the condition (the return value of <i>using_expression</i> is <b>TRUE</b>) can be queried. The operations that are affected include <b>SELECT</b>, <b>UPDATE.... RETURNING</b>, and <b>DELETE... RETURNING</b>.</p> <p>If <i>command</i> is set to <b>UPDATE</b>, only tuple data that meets the condition (the return value of <i>using_expression</i> is <b>TRUE</b>) can be updated. The operations that are affected include <b>UPDATE</b>, <b>UPDATE ... RETURNING</b>, and <b>SELECT ... FOR UPDATE/SHARE</b>.</p> <p>If <i>command</i> is set to <b>DELETE</b>, only tuple data that meets the condition (the return value of <i>using_expression</i> is <b>TRUE</b>) can be deleted. The operations that are affected include <b>DELETE</b> and <b>DELETE ... RETURNING</b>.</p> <p>The following table <a href="#">Table 21-239</a> describes the relationship between row-level access control policies and SQL statements.</p> |
| role_name | Specifies database users affected by a row-level access control policy.<br><b>NOTICE</b><br>System administrators are not affected by row access control.                                                                                                                                          | If this parameter is not specified, the default value <b>PUBLIC</b> will be used, indicating that all database users will be affected. You can specify multiple affected database users.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

| Parameter        | Description                                                                                   | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------|-----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| using_expression | Specifies an expression defined for a row-level access control policy (return type: boolean). | The expression cannot contain aggregate functions and window functions. In the statement rewriting phase of a query, if row-level access control for a data table is enabled, the expressions that meet the specified conditions will be added to the plan tree. The expression is calculated for each tuple in the data table. For <b>SELECT</b> , <b>UPDATE</b> , and <b>DELETE</b> , row data is visible to the current user only when the return value of the expression is <b>TRUE</b> . If the expression returns <b>FALSE</b> , the tuple is invisible to the current user. In this case, the user cannot view the tuple through the <b>SELECT</b> statement, update the tuple through the <b>UPDATE</b> statement, or delete the tuple through the <b>DELETE</b> statement. |

**Table 21-239** Relationship between row-level security policies and SQL statements

| Command                 | SELECT/ALL Policy | UPDATE/ALL Policy | DELETE/ALL Policy |
|-------------------------|-------------------|-------------------|-------------------|
| SELECT                  | Existing row      | No                | No                |
| SELECT FOR UPDATE/SHARE | Existing row      | Existing row      | No                |
| UPDATE                  | No                | Existing row      | No                |
| UPDATE RETURNING        | Existing row      | Existing row      | No                |
| DELETE                  | No                | No                | Existing row      |
| DELETE RETURNING        | Existing row      | No                | Existing row      |

## Example 1: Create a Row-level Access Control Policy That the Current User Can Only View Its Own Data.

1. Create users **alice** and **bob**.  

```
CREATE ROLE alice PASSWORD '{password}';
CREATE ROLE bob PASSWORD '{password}';
```
2. Create the data table **public.all\_data**.  

```
CREATE TABLE public.all_data(id int, role varchar(100), data varchar(100));
```
3. Insert data into the table.  

```
INSERT INTO all_data VALUES(1, 'alice', 'alice data');
INSERT INTO all_data VALUES(2, 'bob', 'bob data');
INSERT INTO all_data VALUES(3, 'peter', 'peter data');
```
4. Grant the read permission for the **all\_data** table to users **alice** and **bob**.  

```
GRANT SELECT ON all_data TO alice, bob;
```
5. Enable row-level access control.  

```
ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;
```
6. Create a row-level access control policy to specify that the current user can view only its own data.  

```
CREATE ROW LEVEL SECURITY POLICY all_data_rls ON all_data USING(role = CURRENT_USER);
```
7. View information about the **all\_data** table.

```
\d+ all_data
```

```
Table "public.all_data"
Column | Type | Modifiers | Storage | Stats target | Description
-----+-----+-----+-----+-----+-----
id | integer | | plain | |
role | character varying(100) | | extended | |
data | character varying(100) | | extended | |
```

```
Row Level Security Policies:
```

```
 POLICY "all_data_rls"
 USING (((role)::name = "current_user"()))
```

```
Has OIDs: no
```

```
Distribute By: HASH(id)
```

```
Location Nodes: ALL DATANODES
```

```
Options: orientation=row, compression=no, enable_rowsecurity=true
```

8. Run **SELECT**.  

```
SELECT * FROM all_data;
id | role | data
---+---+---
1 | alice | alice data
2 | bob | bob data
3 | peter | peter data
(3 rows)
EXPLAIN(COSTS OFF) SELECT * FROM all_data;
QUERY PLAN

Streaming (type: GATHER)
Node/s: All datanodes
-> Seq Scan on all_data
(3 rows)
```

9. Switch to the **alice** user.

```
set role alice password '{password}';
```

10. Perform the **SELECT** operation.  

```
SELECT * FROM all_data;
id | role | data
---+---+---
1 | alice | alice data
(1 row)
EXPLAIN(COSTS OFF) SELECT * FROM all_data;
QUERY PLAN

Streaming (type: GATHER)
```

```
Node/s: All datanodes
-> Seq Scan on all_data
 Filter: ((role)::name = 'alice'::name)
Notice: This query is influenced by row level security feature
(5 rows)
```

## Example 2: Partition Permission Management Through Row-Level Control

1. Create user **alice**.  

```
CREATE ROLE alice PASSWORD '{password1}';
```
2. Create the range partitioned table **web\_returns\_p1** and insert data into the table.  

```
CREATE TABLE web_returns_p1
(
 wr_returned_date_sk integer,
 wr_returned_time_sk integer,
 wr_item_sk integer NOT NULL,
 wr_refunded_customer_sk integer
)
WITH (orientation = column)
DISTRIBUTE BY HASH (wr_item_sk)
PARTITION BY RANGE(wr_returned_date_sk)
(
 PARTITION p2016 START(800) END(830) EVERY(1)
);

INSERT INTO web_returns_p1 values (801,17,11,102);
INSERT INTO web_returns_p1 values (802,18,12,103);
```
3. Grant the read permission on the **web\_returns\_p1** table to user **alice**.  

```
GRANT SELECT ON web_returns_p1 TO alice;
```
4. Enable row-level access control.  

```
ALTER TABLE web_returns_p1 ENABLE ROW LEVEL SECURITY;
```
5. Create row-level access control policy **web\_returns\_rsl**. In the command, **wr\_returned\_date\_sk** is a partition name of the **web\_returns\_p1** partition table, and **801** is the partition value.  

```
CREATE ROW LEVEL SECURITY POLICY web_returns_rsl ON web_returns_p1
USING('wr_returned_date_sk' = '801');
```
6. Impose the row-level access control policy **web\_returns\_rsl** on user **alice**.  

```
ALTER ROW LEVEL SECURITY POLICY web_returns_rsl ON web_returns_p1 TO alice;
```
7. Switch to the **alice** user.  

```
set role alice password '{password1}';
```
8. Query the **web\_returns\_p1** table.  

```
select * from web_returns_p1;
```

## Helpful Links

[ALTER ROW LEVEL SECURITY POLICY, DROP ROW LEVEL SECURITY POLICY](#)

### 21.13.8.2 ALTER ROW LEVEL SECURITY POLICY

#### Function

**ALTER ROW LEVEL SECURITY POLICY** modifies an existing row-level access control policy, including the policy name and the users and expressions affected by the policy.



## Precautions

Only the table owner or administrators can perform this operation.

## Syntax

```
ALTER [ROW LEVEL SECURITY] POLICY [IF EXISTS] policy_name ON table_name RENAME TO
new_policy_name
```

```
ALTER [ROW LEVEL SECURITY] POLICY policy_name ON table_name
[TO { role_name | PUBLIC } [, ...]]
[USING (using_expression)]
```

## Parameter Description

**Table 21-240** ALTER ROW LEVEL SECURITY POLICY parameters

| Parameter        | Description                                                                                                                                                                     | Value Range                                          |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| policy_name      | Specifies the name of a row-level access control policy to be modified.                                                                                                         | A string compliant with the identifier naming rules. |
| table_name       | Specifies the name of a table to which a row-level access control policy is applied.                                                                                            | Name of an existing table.                           |
| new_policy_name  | Specifies the new name of a row-level access control policy.                                                                                                                    | -                                                    |
| role_name        | Specifies names of users affected by a row-level access control policy will be applied. <b>PUBLIC</b> indicates that the row-level access control policy will affect all users. | Name of an existing role.                            |
| using_expression | Specifies an expression defined for a row-level access control policy. The return value is of the boolean type.                                                                 | -                                                    |

## Examples

Enable row-level access control.

```
ALTER TABLE all_data ENABLE ROW LEVEL SECURITY;
```

Change the name of the **all\_data\_rls** policy.

```
ALTER ROW LEVEL SECURITY POLICY all_data_rls ON all_data RENAME TO all_data_new_rls;
```

Change the users affected by the row-level access control policy.

```
ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data TO alice, bob;
```

Modify the expression defined for the access control policy.

```
ALTER ROW LEVEL SECURITY POLICY all_data_new_rls ON all_data USING (id > 100 AND role =
current_user);
```

## Helpful Links

[CREATE ROW LEVEL SECURITY POLICY](#) and [DROP ROW LEVEL SECURITY POLICY](#)

### 21.13.8.3 DROP ROW LEVEL SECURITY POLICY

#### Function

Deletes a row-level access control policy from a table.

#### Precautions

Only the table owner or administrators can delete a row-level access control policy from the table.

#### Syntax

```
DROP [ROW LEVEL SECURITY] POLICY [IF EXISTS] policy_name ON table_name [CASCADE | RESTRICT]
```

#### Parameter Description

**Table 21-241** DROP ROW LEVEL SECURITY POLICY parameters

| Parameter            | Description                                                                                                                                                           | Value Range                                                                            |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| IF EXISTS            | Reports a notice instead of an error if the specified row-level access control policy does not exist.                                                                 | -                                                                                      |
| policy_name          | Specifies the name of a row-level access control policy to be deleted.                                                                                                | An existing name of a row-level access control policy.                                 |
| table_name           | Specifies the name of a table to which a row-level access control policy to be deleted belongs.                                                                       | An existing name of the data table to which a row-level access control policy belongs. |
| CASCADE/<br>RESTRICT | The two parameters are used only for syntax compatibility. No objects depend on access control policies and thereby <b>CASCADE</b> is equivalent to <b>RESTRICT</b> . | -                                                                                      |

#### Examples

Delete the row-level access control policy **all\_data\_rls** from table **all\_data**.

```
DROP ROW LEVEL SECURITY POLICY all_data_rls ON all_data;
```

## Helpful Links

[ALTER ROW LEVEL SECURITY POLICY](#), [CREATE ROW LEVEL SECURITY POLICY](#)

### 21.13.8.4 CREATE REDACTION POLICY

#### Function

**CREATE REDACTION POLICY** creates a data redaction policy for a table.

#### Precautions

- The data masking feature is controlled by **feature\_support\_options**. If error information "REDACTION POLICY is not yet supported, please add enable\_data\_redaction into feature\_support\_options." or "Cannot use the feature of data redaction, please drop existed redaction policy or add enable\_data\_redaction into feature\_support\_options." is displayed, the data masking feature is not enabled. In this case, you need to contact technical support to set **feature\_support\_options**.
- Only the table object owner and users who have been granted the **gs\_redaction\_policy** preset role are authorized to create data masking policies.
- Data masking policies can be created for regular tables and non-EXTERNAL-SCHEMA foreign tables (such as OBS, HDFS, GDS, and foreign tables for coordinated analysis), but not for system catalogs, EXTERNAL SCHEMA foreign tables, temporary tables, UNLOGGED tables, views, or functions.
- Synonyms cannot be used to create redaction policies for ordinary table objects.
- Table objects and redaction policies have a one-to-one mapping relationship. A redaction policy is a collection of data redaction functions that can be applied to multiple columns in a table. You can set different redaction functions for different columns.
- A redaction policy is enabled by default upon its creation, that is, the **enable** parameter of the policy is **true** by default.
- Redaction policies do not take effect on users with the **sysadmin** permission. Data in the redacted columns is always visible to such users.
- Data redaction policies can be matched with specified roles.

#### Syntax

```
CREATE REDACTION POLICY [IF NOT EXISTS] policy_name ON table_name [{ BEFORE | AFTER }
old_policy_name]
[WHEN (when_expression)]
[ADD COLUMN column_name WITH redaction_function_name ([argument [, ...]])] [, ...];
```

## Parameter Description

**Table 21-242** CREATE REDACTION POLICY parameters

| Parameter      | Description                                                                                                                                                                                                                                                                          | Value Range |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| policy_name    | Specifies the name of a redaction policy.                                                                                                                                                                                                                                            | -           |
| table_name     | Specifies the name of the table to which the redaction policy is applied.                                                                                                                                                                                                            | -           |
| BEFORE   AFTER | Specifies the relative location where the current policy is created. Generally, one masking policy is used for one table. The default value indicates that the current policy is created after the last candidate policy of the target table recorded in the current system catalog. | -           |

| Parameter                   | Description                                                                                                                                 | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WHEN<br>( when_expression ) | Specifies the expression used for the redaction policy to take effect. The redaction policy takes effect only when this expression is true. | <p>When a query statement is querying a table where a redaction policy is enabled, the redacted data is invisible in the query only if the WHEN expression for the redaction policy is true. Generally, the WHEN clause is used to specify the users for which the redaction policy takes effect.</p> <p>The WHEN clause must comply with the following rules:</p> <ol style="list-style-type: none"><li>1. The expression can be a combination of multiple subexpressions connected by AND and/or OR.</li><li>2. Each subexpression supports only the =, &lt;&gt;, !=, &gt;=, &gt;, &lt;=, and &lt; operators. The left and right operand values can only be constant values or one of the following system constant values:<br/><b>SESSION_USER</b>,<br/><b>CURRENT_USER</b>, <b>USER</b>,<br/><b>CURRENT_ROLE</b>, and<br/><b>CURRENT_SCHEMA</b> system constants or the <b>SYS_CONTEXT</b> system function.</li><li>3. Each subexpression can be an IN or NOT IN expression. The value for the left operand can be any of the system constant values listed in rule 2, and each element in the array of the right operand must be a constant value.</li><li>4. Each subexpression can be a <b>pg_has_role(user, role, privilege)</b> system function.</li><li>5. If you want a redaction policy to be valid in all conditions, that is, you want it to take effect on all users (including the table owner), you are advised to use the</li></ol> |

| Parameter               | Description                                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                                                                                                                                                               |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         |                                                                                                                                                                                                                                                                                                                    | (1=1) expression to create this policy.<br><br>6. If the WHEN clause is not specified, the redaction policy is disabled by default. You need to manually specify a WHEN expression for the policy to take effect.                                                                                                                         |
| column_name             | Specifies the name of the table column to which the redaction policy is applied.                                                                                                                                                                                                                                   | -                                                                                                                                                                                                                                                                                                                                         |
| redaction_function_name | Specifies the redaction function applied to the specified table column.                                                                                                                                                                                                                                            | -                                                                                                                                                                                                                                                                                                                                         |
| arguments               | Specifies the list of arguments of the redaction function.<br><br>You can use the built-in masking functions <b>MASK_NONE</b> , <b>MASK_FULL</b> , and <b>MASK_PARTIAL</b> , or create your own masking functions by using the C language or PL/pgSQL. For details, see <a href="#">Data Redaction Functions</a> . | <ul style="list-style-type: none"><li>• <b>MASK_NONE</b>: indicates that no masking is performed.</li><li>• <b>MASK_FULL</b>: indicates that all data is masked to a fixed value.</li><li>• <b>MASK_PARTIAL</b>: indicates that partial masking is performed based on the specified character type, numeric type, or time type.</li></ul> |

## Examples

### Create redaction policy for a specified user.

1. Create users **alice** and **matu**.  

```
CREATE ROLE alice PASSWORD '{password}';
CREATE ROLE matu PASSWORD '{password}';
```
2. Create the table object **emp** as user **alice**, and insert data into the table.  

```
CREATE TABLE emp(id int, name varchar(20), salary NUMERIC(10,2));
INSERT INTO emp VALUES(1, 'July', 1230.10), (2, 'David', 999.99);
```
3. Create the redaction policy **mask\_emp** for the **emp** table as user **alice** and make the **salary** column invisible to user **matu**.  

```
CREATE REDACTION POLICY mask_emp ON emp WHEN(current_user = 'matu') ADD COLUMN salary
WITH mask_full(salary);
```
4. Grant the **SELECT** permission on the **emp** table to user **matu** as user **alice**.  

```
GRANT SELECT ON emp TO matu;
```
5. Switch to user **matu**.  

```
SET ROLE matu PASSWORD '{password}';
```

6. Query the **emp** table. Data in the **salary** column has been redacted.  

```
SELECT * FROM emp;
```

#### Create redaction policy for the role.

1. Create the role **redact\_role**.  

```
CREATE ROLE redact_role PASSWORD '{password}';
```
2. Add users **matu** and **alice** to the role **redact\_role**.  

```
GRANT redact_role to matu,alice;
```
3. Create a table object **emp1** as the administrator and insert data.  

```
CREATE TABLE emp1(id int, name varchar(20), salary NUMERIC(10,2));
INSERT INTO emp1 VALUES(3, 'Rose', 2230.20), (4, 'Jack', 899.88);
```
4. Create a redaction policy **mask\_emp1** for the table object **emp1** as the administrator to make the **salary** column invisible to role **redact\_role**.  

```
CREATE REDACTION POLICY mask_emp1 ON emp1 WHEN (pg_has_role(current_user, 'redact_role',
'member')) ADD COLUMN salary WITH mask_full(salary);
```

If no user is specified, the current user (**current\_user**) is used by default.

```
CREATE REDACTION POLICY mask_emp1 ON emp1 WHEN (pg_has_role('redact_role', 'member'))
ADD COLUMN salary WITH mask_full(salary);
```
5. Grant the **SELECT** permission on the **emp1** table to user **matu** as the administrator.  

```
GRANT SELECT ON emp1 TO matu;
```
6. Switch to user **matu**.  

```
SET ROLE matu PASSWORD '{password}';
```
7. Query the **emp** table. Data in the **salary** column has been redacted.  

```
SELECT * FROM emp1;
```

## Helpful Links

[ALTER REDACTION POLICY](#), [DROP REDACTION POLICY](#)

### 21.13.8.5 ALTER REDACTION POLICY

#### Function

**ALTER REDACTION POLICY** modifies a data redaction policy applied to a specified table.

#### Precautions

Only the table object owner and users with the **gs\_redaction\_policy** preset role can modify the masking policy.

#### Syntax

- Modify the expression used for a redaction policy to take effect.  

```
ALTER REDACTION POLICY policy_name ON table_name WHEN (when_expression);
```
- Enable or disable a redaction policy.  

```
ALTER REDACTION POLICY policy_name ON table_name ENABLE | DISABLE;
```
- Rename a redaction policy.  

```
ALTER REDACTION POLICY policy_name ON table_name RENAME TO new_policy_name;
```
- Add, modify, or delete a column on which the redaction policy is used.  

```
ALTER REDACTION POLICY policy_name ON table_name
action;
```

There are several clauses of **action**:

```
[INHERIT] ADD COLUMN column_name WITH redaction_function_name ([argument [, ...]])
| [INHERIT] MODIFY COLUMN column_name WITH redaction_function_name ([argument [, ...]])
| DROP COLUMN column_name
```

## Parameter Description

**Table 21-243** ALTER REDACTION POLICY parameters

| Parameter               | Description                                                                                                                                                                                                                                                                                                                                                   | Value Range                                                                               |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| policy_name             | Specifies the name of the redaction policy to be modified.                                                                                                                                                                                                                                                                                                    | Name of an existing masking policy.                                                       |
| table_name              | Specifies the name of the table to which the redaction policy is applied.                                                                                                                                                                                                                                                                                     | Name of an existing table.                                                                |
| when_expression         | Specifies the new expression used for the redaction policy to take effect.                                                                                                                                                                                                                                                                                    | -                                                                                         |
| ENABLE   DISABLE        | Specifies whether to enable or disable the current redaction policy. <ul style="list-style-type: none"><li>• <b>ENABLE</b>: makes the data masking policy of the table take effect again.</li><li>• <b>DISABLE</b>: disables the masking policy applied to the table.</li></ul>                                                                               | -                                                                                         |
| new_policy_name         | Specifies the new name of the redaction policy.                                                                                                                                                                                                                                                                                                               | A string compliant with the <a href="#">identifier naming rules</a> .                     |
| column_name             | Specifies the name of the table column to which the redaction policy is applied. <ul style="list-style-type: none"><li>• To add a column, use a column name that has not been bound to any redaction functions.</li><li>• To modify a column, use the name of an existing column.</li><li>• To delete a column, use the name of an existing column.</li></ul> | -                                                                                         |
| redaction_function_name | Specifies the name of a redaction function.                                                                                                                                                                                                                                                                                                                   | For details about the supported functions, see <a href="#">Data Redaction Functions</a> . |



| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                     | Value Range                                                                                     |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| arguments | <p>Specifies the list of arguments of the masking function.</p> <ul style="list-style-type: none"><li>• <b>MASK_NONE</b>: indicates that no masking is performed.</li><li>• <b>MASK_FULL</b>: indicates that all data is masked to a fixed value.</li><li>• <b>MASK_PARTIAL</b>: indicates that data of the specified character type, numeric type, or time type is partially masked.</li></ul> | <p>For details about the supported functions, see <a href="#">Data Redaction Functions</a>.</p> |

## Examples

Create a user named **test\_role** and an example table named **emp**, and insert data into the table.

```
CREATE ROLE test_role PASSWORD '{Password}';
```

```
DROP TABLE IF EXISTS emp;
CREATE TABLE emp(id int, name varchar(20), salary NUMERIC(10,2));
INSERT INTO emp VALUES(1, 'July', 1230.10), (2, 'David', 999.99);
```

Define a masking policy **mask\_emp** on the **emp** table that hides the **salary** column from the user **test\_role**.

```
CREATE REDACTION POLICY mask_emp ON emp WHEN(current_user = 'test_role') ADD COLUMN salary
WITH mask_full(salary);
```

Modify the expression for the data masking policy to take effect for the specified role. (If no user is specified, the policy takes effect for the current user by default.)

```
ALTER REDACTION POLICY mask_emp ON emp WHEN (pg_has_role(current_user, 'redact_role', 'member'));
ALTER REDACTION POLICY mask_emp ON emp WHEN (pg_has_role('redact_role', 'member'));
```

Modify the expression for the masking policy to take effect for all users.

```
ALTER REDACTION POLICY mask_emp ON emp WHEN (1=1);
```

Disable the masking policy.

```
ALTER REDACTION POLICY mask_emp ON emp DISABLE;
```

Enable the masking policy again.

```
ALTER REDACTION POLICY mask_emp ON emp ENABLE;
```

Change the masking policy name to **mask\_emp\_new**.

```
ALTER REDACTION POLICY mask_emp ON emp RENAME TO mask_emp_new;
```

Add a column with the masking policy used.

```
ALTER REDACTION POLICY mask_emp_new ON emp ADD COLUMN name WITH mask_partial(name, '*', 1,
length(name));
```

Modify the masking policy for the **name** column. Use the **MASK\_FULL** function to redact all data in the **name** column.

```
ALTER REDACTION POLICY mask_emp_new ON emp MODIFY COLUMN name WITH mask_full(name);
```

Delete an existing column where the masking policy is used.

```
ALTER REDACTION POLICY mask_emp_new ON emp DROP COLUMN name;
```

## Helpful Links

[CREATE REDACTION POLICY](#) and [DROP REDACTION POLICY](#)

### 21.13.8.6 DROP REDACTION POLICY

#### Function

**DROP REDACTION POLICY** deletes a data masking policy applied to a specified table.

#### Precautions

Only the table owner and users granted the **gs\_redaction\_policy** preset role have the permission to delete data masking policies.

#### Syntax

```
DROP REDACTION POLICY [IF EXISTS] policy_name ON table_name;
```

#### Parameter Description

**Table 21-244** DROP REDACTION POLICY parameters

| Parameter   | Description                                                                                       | Value Range                                                                |
|-------------|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|
| IF EXISTS   | Sends a notice instead of throwing an error if the redaction policy to be deleted does not exist. | -                                                                          |
| policy_name | Specifies the name of the masking policy to be deleted.                                           | An existing name of the masking policy.                                    |
| table_name  | Specifies the name of the table with the masking policy to be deleted.                            | An existing name of the table to which the data masking policy is applied. |

#### Examples

Delete the masking policy **mask\_emp** from table **emp**.

```
DROP REDACTION POLICY mask_emp ON emp;
```

## Helpful Links

[ALTER REDACTION POLICY](#), [CREATE REDACTION POLICY](#)

### 21.13.9 Resource Pools and Filtering Rules

## 21.13.9.1 CREATE RESOURCE POOL

### Function

**CREATE RESOURCE POOL** creates a resource pool and specifies the Cgroup for the resource pool. The system resources (CPU, memory, I/O, and storage resources) of a database are limited. When multiple types of services (such as data loading, batch analysis, and real-time query) are running at the same time, they may compete for resources and hinder operations. As a result, the throughput decreases and the overall performance deteriorates. Allocating system resources correctly prevents efficiency loss due to poor resource use. DWS offers a resource management feature that lets you create separate resource pools for different needs. These pools keep resources independent from one another.

This syntax is for reference only. You are advised to create a resource pool on the console. For details about resource pools, see "DWS Resource Load Management" in the *User Guide*.

### Precautions

As long as the current user has **CREATE** permission, it can create a resource pool.

### Syntax

```
CREATE RESOURCE POOL pool_name
[WITH ({MEM_PERCENT=pct | CONTROL_GROUP="group_name" | ACTIVE_STATEMENTS=stmt |
MAX_DOP = dop | MEMORY_LIMIT='memory_size' | io_limits=io_limits | io_priority='priority' |
nodegroup='nodegroup_name' | is_foreign = boolean | except_rule='except_rule' | weight=bandwidth_weight
| enable_concurrency_scaling=boolean}[, ...]]);
```

### Parameter Description

**Table 21-245** CREATE RESOURCE POOL parameters

| Parameter | Description                                                                                                                    | Value Range or Example                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| pool_name | Specifies the name of a resource pool.<br><br>The name of a resource pool cannot be same as that of an existing resource pool. | A string, which must comply with the <a href="#">identifier naming convention</a> . |

| Parameter     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Value Range or Example                                                                                                                                                                                                                                                                                                                                                              |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONTROL_GROUP | <p>Specifies the name of a Cgroup.</p> <p>Control groups (Cgroups) are a mechanism provided by the Linux kernel to restrict, record, and isolate physical resources (such as CPU, memory, and I/O resources) used by process groups. Cgroups have strict restrictions on Linux system resources. If a process is added to a Cgroup, it can use only restricted resources. For details about Cgroup principles, see the product manual corresponding to your OS.</p> <p>The <b>DefaultClass Cgroup</b> is preset in the environment, and the Timeshare Cgroup under the <b>DefaultClass</b> is divided into the following four levels: <b>Rush</b>, <b>High</b>, <b>Medium</b>, and <b>Low</b>.</p> <ul style="list-style-type: none"><li>• You can use either double quotation marks ("" ) or single quotation mark (') in the syntax when setting the name of a Cgroup.</li><li>• The value of <b>group_name</b> is case-sensitive.</li><li>• If <b>group_name</b> is not specified, the string "Medium" will be used by default in the syntax, indicating the <b>Medium</b> Timeshare Cgroup under <b>DefaultClass</b>.</li></ul> | <p>Create a default resource pool, and associate it with the <b>Medium</b> Timeshare Workload Cgroup under <b>DefaultClass</b>.<br/><code>CREATE RESOURCE POOL pool1;</code></p> <p>Create a resource pool, and associate it with the <b>High</b> Timeshare Workload Cgroup under <b>DefaultClass</b>.<br/><code>CREATE RESOURCE POOL pool2 WITH (CONTROL_GROUP="High");</code></p> |

| Parameter         | Description                                                                                                                                                                                                                                                                                                                                      | Value Range or Example                                                                                                                                                                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ACTIVE_STATEMENTS | Specifies the maximum number of complex statements that can be concurrently executed in a resource pool. The value is a number ranging from <b>-1</b> to <i>INT_MAX</i> . If this parameter is not specified, the default value <b>10</b> is used.                                                                                               | Create <b>pool3</b> and set the maximum number of complex statements that can be concurrently executed to 5, the memory usage to 50%, and the network weight to 5.<br><pre>CREATE RESOURCE POOL pool3 WITH (ACTIVE_STATEMENTS=5, MEM_PERCENT=50, weight=5);</pre> |
| MAX_DOP           | Specifies the maximum number of simple SQL statements that can be concurrently executed in a resource pool. The value is a number ranging from <b>-1</b> to <i>INT_MAX</i> . The value <b>-1</b> or <b>0</b> indicates that the maximum number of statements is not limited. If the value is not specified, the default value <b>-1</b> is used. | Create <b>pool4</b> and set the maximum number of concurrent simple SQL statements to 10.<br><pre>CREATE RESOURCE POOL pool4 WITH (MAX_DOP=10);</pre>                                                                                                             |
| MEMORY_LIMIT      | Specifies the estimated maximum memory for a resource pool.<br>A string. <ul style="list-style-type: none"><li>• <b>default</b>: The memory limit is half of the resource pool memory.</li><li>• <b>unlimited</b>: no limit.</li><li>• The value is from 1 KB to 2047 GB.</li></ul> If this parameter is not specified, <b>default</b> is used.  | -                                                                                                                                                                                                                                                                 |
| MEM_PERCENT       | Specifies the proportion of available resource pool memory to the total memory or group user memory. <ul style="list-style-type: none"><li>• The value ranges from <b>0</b> to <b>100</b>. The default value is <b>0</b>.</li></ul>                                                                                                              | -                                                                                                                                                                                                                                                                 |

| Parameter   | Description                                                                                                                                                                         | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| io_limits   | This parameter has been discarded in 8.1.2 and is reserved for compatibility with earlier versions.                                                                                 | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| io_priority | This parameter has been discarded in 8.1.2 and is reserved for compatibility with earlier versions.                                                                                 | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| nodegroup   | Specifies the name of a logical cluster where the resource pool is. The logical cluster must already exist.                                                                         | If the logical cluster name contains uppercase letters or special characters or begins with a digit, enclose the name with double quotation marks in SQL statements.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| is_foreign  | In logical cluster mode, the current resource pool is used to control the resources of common users who are not associated with the logical cluster specified by <b>nodegroup</b> . | <ul style="list-style-type: none"><li>• <b>nodegroup</b> must specify an existing logical cluster, and cannot be <b>elastic_group</b> or the default Node Group (<b>group_version1</b>), which is generated during cluster installation.</li><li>• If <b>is_foreign</b> is set to <b>true</b>, the resource pool cannot be associated with users. That is, <b>CREATE USER... RESOURCE POOL</b> cannot be used to configure resource pools for users. The resource pool automatically checks whether the users are associated with its logical cluster. If they are not, they will be controlled by the resource pool when performing operations on DNs in the logical cluster.</li></ul> |

| Parameter                  | Description                                                                                                                                                                                                                                                                                                                                                                                      | Value Range or Example                                                                                                                                                                                                                                                                                                        |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| except_rule                | Name of an exception rule set.<br><br>An exception rule set is a set of rule thresholds that define the job execution time, job queuing time, CPU usage, and maximum disk flushing. It is used together with a resource pool. When an exception rule is triggered during job execution, the job is terminated or degraded.<br><br>For more information, see <a href="#">CREATE EXCEPT RULE</a> . | <a href="#">Example: Create an exception rule to terminate a job when the job runs for more than 10 seconds.</a>                                                                                                                                                                                                              |
| weight                     | Specifies the network bandwidth weight of the resource pool. The value is an integer ranging from <b>-1</b> to <b>2147483647</b> . The default value is <b>-1</b> , indicating that no weight is set.                                                                                                                                                                                            | -                                                                                                                                                                                                                                                                                                                             |
| enable_concurrency_scaling | Specifies whether to enable the elastic concurrency expansion function. This function is supported only by clusters of version 9.1.0.100 or later.                                                                                                                                                                                                                                               | <ul style="list-style-type: none"><li>• <b>true</b> indicates that the elastic concurrent expansion function of jobs in the resource pool is enabled.</li><li>• <b>false</b> indicates that the elastic concurrent expansion function of jobs in the resource pool is disabled.</li></ul> The default value is <b>false</b> . |

## Examples

This example assumes that Cgroups have been created by users in advance.

Create a default resource pool, and associate it with the **Medium** Timeshare Workload Cgroup under **DefaultClass**.

```
CREATE RESOURCE POOL pool1;
```

Create a resource pool, and associate it with the **High** Timeshare Workload Cgroup under **DefaultClass**.

```
CREATE RESOURCE POOL pool2 WITH (CONTROL_GROUP="High");
```

Create **pool\_p** and set the maximum number of complex statements that can be concurrently executed to 5, the memory usage to 50%, and the network weight to 5.

```
CREATE RESOURCE POOL pool_p WITH (ACTIVE_STATEMENTS=5, MEM_PERCENT=50, weight=5);
```

Query the configuration parameters of the resource pool in the PG\_RESOURCE\_POOL system catalog.

```
SELECT * FROM PG_RESOURCE_POOL WHERE respool_name = 'pool_p';
```

```
gaussdb-> SELECT * FROM PG_RESOURCE_POOL WHERE respool_name = 'pool_p';
respool_name | mem_percent | cpu_affinity | control_group | active_statements | max_dop | memory_limit | parentid | io_li
mits | io_priority | nodegroup | is_foreign | short_acc | except_rule | weight | enable_concurrency_scaling

pool_p | 50 | -1 | DefaultClass:Medium | 5 | -1 | default | | 0
(1 row)
```

Create an exception rule set **except\_rule4**, set **elapsedtime** to **10** (unit: seconds), and create a resource pool and bind the exception rule to the resource pool.

```
CREATE EXCEPT RULE except_rule4 WITH (elapsedtime=10);
CREATE resource pool resource_pool_a4 WITH (except_rule='except_rule4');
```

Create a resource pool and enable the elastic concurrent expansion function of the resource pool.

```
CREATE RESOURCE POOL pool6 WITH (enable_concurrency_scaling=true);
```

## Example: Create an exception rule to terminate a job when the job runs for more than 10 seconds.

In DWS resource load management, exception rules take effect only after being bound to a resource pool, and then the user who executes a job is bound to a specified resource pool. If an exception rule is triggered when a user executes a job, the job will be terminated. Perform the following operations to simulate the termination process.

### Step 1 Create a test table and import data to the table.

1. Connect to the database as the system administrator and run the following SQL statements to create temporary table **src** to provide data sources for subsequent INSERT operations:
 

```
CREATE SCHEMA test;
CREATE TABLE test.src AS SELECT 1;
```
2. Run the following SQL statements to create table **test.test**:
 

```
DROP TABLE IF EXISTS test.test;
CREATE TABLE test.test(a int, b numeric(15,2)) WITH(orientation=column);
```
3. Run the SQL statements to insert 20 million rows of data into the two tables. It takes about half a minute. Please wait.
 

```
INSERT INTO test.test SELECT generate_series(1,20000000) % 1000,generate_series(1,20000000)
FROM test.src;
```

### Step 2 Create exception rule set **except\_rule4** and set **elapsedtime** to 10 seconds.

```
CREATE EXCEPT RULE except_rule4 WITH (elapsedtime=10);
```

### Step 3 Create resource pool **resource\_pool\_a4** and bind the exception rule to it.

```
CREATE resource pool resource_pool_a4 WITH (except_rule='except_rule4');
```

### Step 4 Create a job user and bind the user to **resource\_pool\_a4**. The password is configured as required.

```
CREATE USER user4 RESOURCE POOL 'resource_pool_a4' PASSWORD 'password';
```



**Step 5** Grant the permission to access table **test.test** to **user4**.

```
GRANT USAGE ON SCHEMA test TO user4;
GRANT SELECT ON TABLE test.test TO user4;
```

**Step 6** Connect to the database as **user4** and run the following SQL statement. Once the statement is executed for more than 10 seconds, it is terminated.

```
SELECT * FROM test.test;
```

```
gaussdb=> SELECT * FROM test.test;
ERROR: canceling statement due to workload manager exception.
DETAIL: except rule [except_rule4] is meet condition:rule [elapsedtime] is over limit, current value is:11.
gaussdb=>
```

**Step 7** Check exception rule definitions in the **PG\_EXCEPT\_RULE** system catalog.

```
SELECT * FROM PG_EXCEPT_RULE WHERE name = 'except_rule4';
```

```
gaussdb=> SELECT * FROM PG_EXCEPT_RULE WHERE name = 'except_rule4';
 name | rule | value
-----+-----+-----
except_rule4 | action | abort
except_rule4 | elapsedtime | 10
(2 rows)
```

----End

## Helpful Links

[ALTER RESOURCE POOL](#), [DROP RESOURCE POOL](#), [CREATE EXCEPT RULE](#)

### 21.13.9.2 ALTER RESOURCE POOL

#### Function

**ALTER RESOURCE POOL** changes the Cgroup of a resource pool.

#### Precautions

Users having the ALTER permission can modify resource pools.

#### Syntax

```
ALTER RESOURCE POOL pool_name
 WITH ({MEM_PERCENT=pct | CONTROL_GROUP='group_name' | ACTIVE_STATEMENTS=stmt | MAX_DOP
= dop | MEMORY_LIMIT='memory_size' | io_limits=io_limits | io_priority='priority' |
nodegroup='nodegroup_name' | except_rule='except_rule' | weight=bandwidth_weight |
enable_concurrency_scaling=boolean}[, ...]);
```

## Parameter Description

**Table 21-246** ALTER RESOURCE POOL parameters

| Parameter   | Description                                                                                                             | Value Range                                                                                                                                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pool_name   | Specifies the name of the resource pool to be modified.                                                                 | Name of an existing resource pool                                                                                                                                                                                           |
| group_name  | Specifies the name of a Cgroup.                                                                                         | Name of an existing Cgroup                                                                                                                                                                                                  |
| stmt        | Specifies the maximum number of statements that can be concurrently executed in a resource pool.                        | Numeral type. The value ranges from -1 to INT_MAX.                                                                                                                                                                          |
| dop         | Specifies the maximum number of simple SQL statements that can be concurrently executed in a resource pool.             | Numeral type. The value ranges from -1 to INT_MAX.                                                                                                                                                                          |
| memory_size | Specifies the estimated maximum memory for a resource pool.                                                             | Value range: a string <ul style="list-style-type: none"><li>● <b>default</b>: The memory limit is half of the resource pool memory.</li><li>● <b>unlimited</b>: no limit.</li><li>● Value range: 1 KB to 2047 GB.</li></ul> |
| mem_percent | Specifies the proportion of available resource pool memory to the total memory or group user memory.                    | The value of <b>mem_percent</b> for a common user is an integer ranging from 0 to 100. The default value is <b>0</b> .                                                                                                      |
| io_limits   | This parameter has been discarded in the 8.1.2 cluster version and is reserved for compatibility with earlier versions. | -                                                                                                                                                                                                                           |
| io_priority | This parameter has been discarded in the 8.1.2 cluster version and is reserved for compatibility with earlier versions. | -                                                                                                                                                                                                                           |
| except_rule | Specifies the exception rule set.                                                                                       | -                                                                                                                                                                                                                           |
| weight      | Specifies the network bandwidth weight of a resource pool.                                                              | -                                                                                                                                                                                                                           |

| Parameter                  | Description                                                                                                                                                                            | Value Range                                                                                                                                                                                                                                                                                                           |
|----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enable_concurrency_scaling | Specifies whether to enable the elastic concurrency expansion function. This function is supported only by clusters of version 9.1.0.100 or later. The default value is <b>false</b> . | <b>true</b> or <b>false</b> <ul style="list-style-type: none"><li>• <b>true</b> indicates that the elastic concurrent expansion function of jobs in the resource pool is enabled.</li><li>• <b>false</b> indicates that the elastic concurrent expansion function of jobs in the resource pool is disabled.</li></ul> |

 **NOTE**

- You can use double quotation marks (") or single quotation marks (') to set the Cgroup name **group\_name**.
- The value of **group\_name** is case-sensitive.
- When **group\_name** is not specified, the default value "Medium" is used. It is the "Medium" Timeshare Cgroup of the DefaultClass Cgroup.
- If a database user specifies the Timeshare string (**Rush**, **High**, **Medium**, or **Low**) in the syntax, for example, if **control\_group** is set to **High**, the resource pool will be associated with the **High** Timeshare Cgroup under **DefaultClass**.

## Examples

Create an example resource pool **pool\_test**, whose Cgroup is **Medium Timeshare Workload** under **DefaultClass**.

```
DROP RESOURCE POOL IF EXISTS pool_test;
CREATE RESOURCE POOL pool_test;
```

Specify **High Timeshare Workload** under **DefaultClass** as the Cgroup for the resource pool **pool\_test**.

```
ALTER RESOURCE POOL pool_test WITH (CONTROL_GROUP="High");
```

Disable the elastic concurrent expansion function for jobs in a specified resource pool.

```
ALTER RESOURCE POOL pool_test WITH (enable_concurrency_scaling=false);
```

## Helpful Links

[CREATE RESOURCE POOL](#) and [DROP RESOURCE POOL](#)

### 21.13.9.3 DROP RESOURCE POOL

#### Function

**DROP RESOURCE POOL** deletes a resource pool.

 **NOTE**

If a role has been associated with a resource pool, the resource pool cannot be deleted.

## Precautions

The user must have the DROP permission in order to delete a resource pool.

## Syntax

```
DROP RESOURCE POOL [IF EXISTS] pool_name;
```

## Parameter Description

**Table 21-247** DROP RESOURCE POOL parameters

| Parameter | Description                                                                       | Value Range                                             |
|-----------|-----------------------------------------------------------------------------------|---------------------------------------------------------|
| IF EXISTS | Sends a notice instead of an error if the specified resource pool does not exist. | -                                                       |
| pool_name | Specifies the name of a created resource pool.                                    | A string, which must comply with the naming convention. |

## Example

Delete resource pool **pool**.

```
DROP RESOURCE POOL pool;
```

## Helpful Links

[ALTER RESOURCE POOL](#), [CREATE RESOURCE POOL](#)

### 21.13.9.4 CREATE EXCEPT RULE

## Function

**CREATE EXCEPT RULE** creates an exception rule set. You can specify specific rule thresholds and the operations (**abort** or **penalty**) to be performed after an exception rule is triggered.

In a distributed database, query tasks are executed by multiple nodes in parallel. Each node independently processes part of the data and summarizes the results. However, if any node encounters a performance bottleneck, the entire query will be slowed down even if other nodes complete the query quickly. In some extreme cases, a query statement may occupy a large number of resources, causing high node resource usage. As a result, compute or storage resources are insufficient, affecting services and even causing performance deterioration or even unavailability of the entire cluster.

When configuring exception rules in DWS, you can set resource restrictions (such as execution time, memory, and disk) for queries. When an exception rule is triggered during SQL statement execution, the execution is automatically terminated. This prevents abnormal queries from breaking down the cluster, ensuring stability and reducing the O&M workload of the database administrator. For example, if the execution time, disk space, or memory usage exceeds the threshold, the query is automatically terminated.

In DWS resource load management, exception rules take effect only after being bound to a resource pool, and then the user who executes a job is bound to a specified resource pool. If an exception rule is triggered when a user executes a job, the job will be terminated.

You can also set resource pools and exception rules on the management console. For details, see "DWS Resource Management Overview" in the *User Guide*.

## Precautions

Exception rules within the same group work together using an AND condition. For example, if the execution time is set to 1,000 seconds (setting **elapsetime** to **1000**) and the memory limit is set to 500 MB (setting **memsize** to **500**), the system terminates a job only when both conditions are met. Reaching just one threshold does not stop the job.

## Syntax

```
CREATE EXCEPT RULE except_rule_name
WITH (
 | BLOCKTIME = VALUE,
 | CPUTIME = VALUE,
 | ELAPSETIME = VALUE,
 | CPUSKEWPERCENT = VALUE,
 | SPILLSIZE = VALUE,
 | BROADCASTSIZE = VALUE,
 | MEMSIZE = VALUE,
 | CPUAVGPERCENT = VALUE,
 | BANDWIDTH = VALUE,
 | ACTION = ['abort' | 'penalty']
);
```

## Parameter Description

Table 21-248 CREATE EXCEPT RULE parameters

| Parameter        | Description                                         | Value Range                                                                                                                                                        |
|------------------|-----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| except_rule_name | Name of an exception rule set.                      | A string, which must comply with <a href="#">Identifier Naming Conventions</a> .                                                                                   |
| blocktime        | Maximum duration of job queue blocking, in seconds. | Numeric type. The value can be <b>-1</b> or ranges from <b>1</b> to <b>INT64_MAX</b> . Set the value to <b>-1</b> , which indicates that the value is not limited. |

| Parameter              | Description                                                                 | Value Range                                                              |
|------------------------|-----------------------------------------------------------------------------|--------------------------------------------------------------------------|
| elapsedtime            | Maximum job execution duration, in seconds.                                 | Numeric type. The value can be -1 or ranges from 1 to <b>INT64_MAX</b> . |
| allcputime             | Maximum CPU time used during job running. The unit is second.               | Numeric type. The value can be -1 or ranges from 1 to <b>INT64_MAX</b> . |
| cpuske<br>wpercen<br>t | Average CPU usage during job execution. The unit is percentage.             | Numeric type. The value can be -1 or ranges from 1 to 100.               |
| cpuavgp<br>ercent      | CPU usage skew during job execution. The unit is percentage.                | Numeric type. The value can be -1 or ranges from 1 to 100.               |
| spillsize              | Maximum size of data spilled to disks during job execution. The unit is MB. | Numeric type. The value can be -1 or ranges from 1 to <b>INT64_MAX</b> . |
| broadca<br>stsize      | Maximum broadcast size of a job. The unit is MB.                            | Numeric type. The value can be -1 or ranges from 1 to <b>INT64_MAX</b> . |
| memsiz<br>e            | Maximum memory size used for job execution. Unit: MB.                       | Numeric type. The value can be -1 or ranges from 1 to <b>INT64_MAX</b> . |
| bandwi<br>dth          | Maximum bandwidth that can be used for job execution. Unit: MB.             | Numeric type. The value can be -1 or ranges from 1 to <b>INT64_MAX</b> . |

## Basic Example

Create an exception rule set (**except\_rule1**) and set the **blocktime** to 3000 seconds and **spillsize** to 4000 MB. If a statement is blocked for more than 3000 seconds and the spill size exceeds 4000 MB, the statement will be terminated.

```
CREATE EXCEPT RULE except_rule1 WITH (blocktime=3000, spillsize=4000, action=abort);
```

Create an exception rule set (**except\_rule2**) and set the **memsize** threshold to 5000 MB. The default operation following an exception is **abort**.

```
CREATE EXCEPT RULE except_rule2 WITH (memsize=3000);
```

Create a resource pool and bind it to exception rule set **except\_rule3**.

```
CREATE resource pool resource_pool_a1 WITH (except_rule='except_rule3');
```

## Example: Create an exception rule to terminate a job when the job runs for more than 10 seconds.

In DWS resource load management, exception rules take effect only after being bound to a resource pool, and then the user who executes a job is bound to a specified resource pool. If an exception rule is triggered when a user executes a job, the job will be terminated. Perform the following operations to simulate the termination process.

### Step 1 Create a test table and import data to the table.

1. Connect to the database as the system administrator and run the following SQL statements to create temporary table **src** to provide data sources for subsequent INSERT operations:  

```
CREATE SCHEMA test;
CREATE TABLE test.src AS SELECT 1;
```
2. Run the following SQL statements to create table **test.test**:  

```
DROP TABLE IF EXISTS test.test;
CREATE TABLE test.test(a int, b numeric(15,2)) WITH(orientation=column);
```
3. Run the SQL statements to insert 20 million rows of data into the two tables. It takes about half a minute. Please wait.  

```
INSERT INTO test.test SELECT generate_series(1,20000000) % 1000,generate_series(1,20000000)
FROM test.src;
```

### Step 2 Create exception rule set **except\_rule4** and set **elapsedtime** to 10 seconds.

```
CREATE EXCEPT RULE except_rule4 WITH (elapsedtime=10);
```

### Step 3 Create resource pool **resource\_pool\_a4** and bind the exception rule to it.

```
CREATE resource pool resource_pool_a4 WITH (except_rule='except_rule4');
```

### Step 4 Create a job user and bind the user to **resource\_pool\_a4**. The password is configured as required.

```
CREATE USER user4 RESOURCE POOL 'resource_pool_a4' PASSWORD 'password';
```

### Step 5 Grant the permission to access table **test.test** to **user4**.

```
GRANT USAGE ON SCHEMA test TO user4;
GRANT SELECT ON TABLE test.test TO user4;
```

### Step 6 Connect to the database as **user4** and run the following SQL statement. Once the statement is executed for more than 10 seconds, it is terminated.

```
SELECT * FROM test.test;
```

```
gaussdb=> SELECT * FROM test.test;
ERROR: canceling statement due to workload manager exception.
DETAIL: except rule [except_rule4] is meet condition:rule [elapsedtime] is over limit, current value is:11.
gaussdb=>
```

### Step 7 Check exception rule definitions in the **PG\_EXCEPT\_RULE** system catalog.

```
SELECT * FROM PG_EXCEPT_RULE WHERE name = 'except_rule4';
```

```
gaussdb=> SELECT * FROM PG_EXCEPT_RULE WHERE name = 'except_rule4';
 name | rule | value
-----+-----+-----
except_rule4 | action | abort
except_rule4 | elapsedtime | 10
(2 rows)
```

----End

Helpful Links

[ALTER EXCEPT RULE](#), [DROP EXCEPT RULE](#), [CREATE RESOURCE POOL](#)

21.13.9.5 ALTER EXCEPT RULE

Function

This syntax modifies an exception rule set. You can modify one or more specific rule thresholds in a rule set.

Precautions

When **ALTER** is used to add a new rule threshold, the original rules in the **WITH** parameter are retained and are not cleared. For example, if a rule set contains only one rule, that is, the maximum duration of job queue blocking (**blocktime**) is 100 seconds, and you want to add another rule, that is, the maximum job execution time (**elapsedtime**) is 1000 seconds, the rule set will contain two rules. For details, see the following example.

Syntax

```
ALTER EXCEPT RULE except_rule_name
 WITH (
 | BLOCKTIME = VALUE,
 | CPUTIME = VALUE,
 | ELAPSEDTIME = VALUE,
 | CPUSKEWPERCENT = VALUE,
 | SPILLSIZE = VALUE,
 | BROADCASTSIZE = VALUE,
 | MEMSIZE = VALUE,
 | CPUAVGPERCENT = VALUE,
 | BANDWIDTH = VALUE,
 | ACTION = ['abort' | 'penalty']
);
```

Parameter Description

Table 21-249 ALTER EXCEPT RULE parameters

| Parameter | Description                                         | Value Range                                                                            |
|-----------|-----------------------------------------------------|----------------------------------------------------------------------------------------|
| rule_name | Name of an exception rule set.                      | A string compliant with the <a href="#">identifier naming rules</a> .                  |
| blocktime | Maximum duration of job queue blocking, in seconds. | Numeric type. The value can be <b>-1</b> or ranges from <b>1</b> to <b>INT64_MAX</b> . |



| Parameter      | Description                                                          | Value Range                                                      |
|----------------|----------------------------------------------------------------------|------------------------------------------------------------------|
| elapsedtime    | Maximum job execution duration, in seconds.                          | Numeric type. The value can be -1 or ranges from 1 to INT64_MAX. |
| allcputime     | Maximum CPU time used during job running, in seconds.                | Numeric type. The value can be -1 or ranges from 1 to INT64_MAX. |
| cpuskewpercent | CPU usage skew during job execution, in percentage.                  | Numeric type. The value can be -1 or ranges from 1 to 100.       |
| cpuavgpercent  | Average CPU usage during job execution, in percentage.               | Numeric type. The value can be -1 or ranges from 1 to 100.       |
| spillsize      | Maximum size of data spilled to disks during job execution, in MB.   | Numeric type. The value can be -1 or ranges from 1 to INT64_MAX. |
| broadcastsize  | Maximum broadcast size of a job, in MB.                              | Numeric type. The value can be -1 or ranges from 1 to INT64_MAX. |
| memsize        | Maximum memory size used for job execution, in MB.                   | Numeric type. The value can be -1 or ranges from 1 to INT64_MAX. |
| bandwidth      | Maximum network bandwidth that can be used for job execution, in MB. | Numeric type. The value can be -1 or ranges from 1 to INT64_MAX. |

## Examples

Create an exception rule **except\_rule1**.

```
CREATE EXCEPT RULE except_rule_1 WITH (blocktime=2000, spillsize=3000, action=abort);
```

Change the blocktime threshold of exception rule set **except\_rule1** to **3000s** and the space for spilled data to **4000 MB**.

```
ALTER EXCEPT RULE except_rule_1 WITH (blocktime=3000, spillsize=4000);
```

Check whether the exception rule is modified successfully by querying the **PG\_EXCEPT\_RULE** system catalog.

```
SELECT * FROM PG_EXCEPT_RULE WHERE name = 'except_rule_1';
```

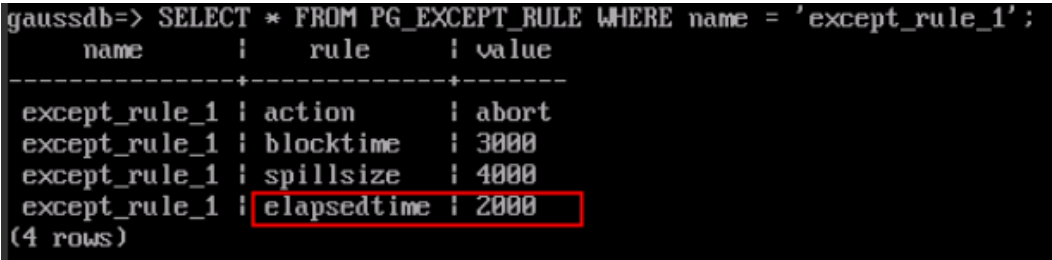
```
gaussdb=> SELECT * FROM PG_EXCEPT_RULE WHERE name = 'except_rule_1';
 name | rule | value
-----+-----+-----
except_rule_1 | action | abort
except_rule_1 | blocktime | 3000
except_rule_1 | spillsize | 4000
(3 rows)
```

Add a new rule, that is, the maximum job execution time (**elapsedtime**) is 2000 seconds.

```
ALTER EXCEPT RULE except_rule_1 WITH (elapsedtime=2000);
```

Check the exception rule again. The rule is modified successfully. **Note that the original rule is retained instead of being cleared.**

```
SELECT * FROM PG_EXCEPT_RULE WHERE name = 'except_rule_1';
```



Create a resource pool and bind it to exception rule set **except\_rule\_1**.

```
CREATE resource pool resource_pool_a1 WITH (except_rule='except_rule_1');
```

Unbind the exception rule set from the resource pool **resource\_pool\_a1**.

```
ALTER resource pool resource_pool_a1 WITH (except_rule='None');
```

Helpful Links

[CREATE EXCEPT RULE](#) and [DROP EXCEPT RULE](#)

21.13.9.6 DROP EXCEPT RULE

Function

This syntax deletes a specified exception rule set.

Precautions

Only the system administrator can perform the **DROP EXCEPT RULE** operation.

Syntax

```
DROP EXCEPT RULE [IF EXISTS]
rule_name ;
```

Parameter Description

Table 21-250 DROP EXCEPT RULE parameters

| Parameter | Description                                                                         | Value Range |
|-----------|-------------------------------------------------------------------------------------|-------------|
| IF EXISTS | Sends a message instead of an error if the specified exception rule does not exist. | -           |

| Parameter | Description                                   | Value Range                             |
|-----------|-----------------------------------------------|-----------------------------------------|
| rule_name | Indicates the name of the exception rule set. | Name of an existing exception rule set. |

## Examples

Create exception rule set **except\_rule\_1**.

```
CREATE EXCEPT RULE except_rule_1 WITH (blocktime=2000, spillsize=3000, action=abort);
```

Delete rule set **except\_rule\_1**.

```
DROP EXCEPT RULE except_rule_1;
```

Delete rule set **except\_rule\_2** if it exists.

```
DROP EXCEPT RULE IF EXISTS except_rule_2;
```

## Links

[CREATE EXCEPT RULE, ALTER EXCEPT RULE](#)

### 21.13.9.7 CREATE BLOCK RULE

## Function

**CREATE BLOCK RULE** creates a query filtering rule, including the query filtering rule name, bound client name, client IP address, user, and matching mode. During routine database operations, executing abnormal SQL statements may use too many resources, affecting database performance and stability. You can create query filtering rules to intercept such SQL statements.

Currently, you can specify the regular expression of an SQL statement for interception. For example, intercepting the SELECT statement of a table of a user. You can also specify the value of **unique\_sql\_id** or **sql\_hash** of an SQL statement for interception.

This document provides basic SQL syntax. You can also configure the query filtering rules on the console. For details, see "Using DWS Query Filters to Intercept Slow SQL Statements" in the *User Guide*.

## Precautions

- This syntax is supported only by clusters of version 9.1.0.100 or later.
- Only a user with the database owner permission or the **gs\_role\_block** role permission can run **CREATE BLOCK RULE**. A system administrator has this permission by default.
- Common users cannot run **CREATE BLOCK RULE**. To run it, they must be authorized by the system administrator. The authorization syntax is as follows:  

```
GRANT gs_role_block TO user;
```

You are advised to narrow down the application scope when creating a query filtering rule to improve performance.

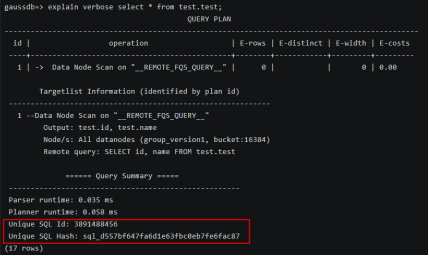
## Syntax

```
CREATE BLOCK RULE [IF NOT EXISTS] block_name
[[TO user_name@'host'] | [TO user_name] | [TO 'host']] |
[FOR UPDATE | SELECT | INSERT | DELETE | MERGE] |
FILTER BY
{ SQL ('text') | TEMPLATE (template_parameter = value) }
[WITH ({ with_parameter = value }, [, ...])];
```

## Parameter Description

Table 21-251 CREATE BLOCK RULE parameters

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| block_name | Specifies the name of the filtering rule to be created.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | A string compliant with the <a href="#">identifier naming rules</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| TO         | <p>Specifies the object or client IP address that determines whether the filtering rule takes effect. The options are as follows:</p> <ul style="list-style-type: none"><li>• <b>TO user_name@'host'</b>: The filtering rule takes effect only when both the username and client IP address are met.</li><li>• <b>TO user_name</b>: The query filtering rule takes effect only when the username is met, regardless of the client IP address.</li><li>• <b>TO 'host'</b>: The query filtering rule takes effect only when the client IP address is met, regardless of the username.</li></ul> | <p>The query filtering rule takes effect only when the client IP address is <b>192.168.0.59</b> and the login username is <b>user1</b>.<br/>CREATE BLOCK RULE query_block TO user1@'192.168.0.59' FOR SELECT FILTER BY SQL('test.test');</p> <p>The query filtering rule takes effect only when the login username is <b>user1</b>, regardless of the client IP address.<br/>CREATE BLOCK RULE query_block TO user1 FOR SELECT FILTER BY SQL('test.test');</p> <p>The query filtering rule takes effect only when the client IP address is <b>192.168.0.59</b>, regardless of the login user.<br/>CREATE BLOCK RULE query_block TO '192.168.0.59' FOR SELECT FILTER BY SQL('test.test');</p> |
| FOR        | Specifies the statement type. Its value can be UPDATE, SELECT, INSERT, DELETE, or MERGE INTO.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FILTER BY | <p>Specifies the filtering method. Value:</p> <ul style="list-style-type: none"><li>• <b>SQL</b>: SQL statements are filtered based on its content. Regular expression matching is performed on statements based on keywords. For example, the length of a table name cannot exceed 1024 characters. It is recommended that the keywords be as simple as possible.</li><li>• <b>TEMPLATE</b>: SQL statements are filtered based on templates. This method intercepts specific statements executed by all users, which is more efficient. The options are as follows:<ul style="list-style-type: none"><li>– <b>unique_sql_id</b>: normalized 64-bit hash value, which is more likely to be repeated than <b>sql_hash</b>.</li><li>– <b>sql_hash</b>: normalized hash value (MD5), which is generally unique and is recommended compared with <b>unique_sql_id</b>.</li></ul></li></ul> | <p>Query the values of <b>unique_sql_id</b> and <b>sql_hash</b> using EXPLAIN VERBOSE.</p> <pre>EXPLAIN VERBOSE SELECT * FROM test.test;</pre>  <pre>gaussdb&gt; explain verbose select * from test.test; QUERY PLAN ----- id   operation   E-rows   E-distinct   E-width   E-cost 1   -&gt; Data Node Scan on "...REMOTE_FQS_QUERY..."   0     0   0.00  Targetlist Information (identified by plan id) ----- 1 -&gt; Data Node Scan on "...REMOTE_FQS_QUERY..." Output: test.id, test.name Node/s: All datanodes (group version1, bucket:16384) Recreate query: SELECT id, name FROM test.test  ===== Query Summary ===== Parser runtime: 0.035 ms Planner runtime: 0.058 ms Unique SQL ID: 309400856 Unique SQL Hash: sql_d5570f6d7f6dd1e3fbc8eb7fedf6c97 (17 rows)</pre> |

| Parameter      | Description                                                | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| with_parameter | Indicates the parameters for querying filter rule details. | <p>The following parameters can be set together. The query that meets any of the parameters will be filtered.</p> <ul style="list-style-type: none"><li>• <b>application_name</b>: client name.</li><li>• <b>query_band</b>: load identifier.</li><li>• <b>table_num</b>: number of base tables scanned by the statement.</li><li>• <b>partition_num</b>: maximum number of partitions to be scanned by the operator.</li><li>• <b>estimate_row</b>: estimated maximum number of table rows scanned by the operator.</li><li>• <b>resource_pool</b>: name of the resource pool to be switched to.</li><li>• <b>max_active_num</b>: maximum number of concurrent statements corresponding to the rule.</li><li>• <b>is_warning</b>: whether an alarm or error is reported when a statement is intercepted. <b>on</b> indicates that an alarm is reported, and <b>off</b> (default value) indicates that an error is reported.</li></ul> |

## Example: Intercept SQL Statements Based on Keywords

Connect to the database as the system administrator and run the following SQL statements to create a temporary table (**src**) to provide data sources for subsequent INSERT operations:

```
CREATE SCHEMA test;
CREATE TABLE test.src AS SELECT 1;
```

Run the following SQL statements to create table **test.test**:

```
DROP TABLE IF EXISTS test.test;
CREATE TABLE test.test(a int, b numeric(15,2)) WITH(orientation=column);
```

Run the SQL statements to insert 20 million rows of data into the two tables. It takes about half a minute. Please wait.

```
INSERT INTO test.test SELECT generate_series(1,20000000) % 1000,generate_series(1,20000000) FROM test.src;
```

Create **user1** and grant the user the permission to access table **test.test**.

```
CREATE USER user1 PASSWORD "password";
GRANT USAGE ON SCHEMA test TO user1;
GRANT SELECT ON TABLE test.test TO user1;
```

Connect to the database as **user1** and query table data. The query result shows that **user1** can query data in table **test.test** without creating a filtering rule

```
SELECT sum(b) FROM test.test GROUP BY a;
```

```
gaussdb=> SELECT sum(b) FROM test.test GROUP BY a;
 sum

199990060000.00
199990120000.00
199990180000.00
199990240000.00
199990300000.00
199990360000.00
199990420000.00
199990480000.00
199990540000.00
199990600000.00
199990660000.00
199990720000.00
199990780000.00
```

Connect to the database as a system administrator and create a query filtering rule (query\_block) to intercept the SQL statements queried by **user1**. The filtering method is **SQL**. If an SQL statement contains the keyword **test** (table name), the SQL statement is filtered.

```
CREATE BLOCK RULE query_block
TO user1@'192.168.0.59' --The rule applies to user1 and the client IP address is 192.168.0.59.
FOR SELECT --Filter SELECT-related operations.
FILTER BY
SQL('test.test') --The filtering mode is SQL. Any statements that contain keyword test (table
name) are intercepted.
WITH --The statements that meet the following conditions are intercepted.
(
 application_name='gsq', --The client name is gsq.
 table_num='1', --The number of base table scans is 1.
 estimate_row='10', --The number of rows in the base table to be scanned is 10.
 is_warning='off' --An error is reported.
)
;
```

Query the created filtering rule by using the **pg\_get\_blockruledef** view. The command output indicates that the rule has taken effect.

```
SELECT * FROM pg_get_blockruledef('query_block');
```

```
gaussdb=> SELECT * FROM pg_get_blockruledef('query_block');
pg_get_blockruledef

CREATE BLOCK RULE query_block TO user1@'192.168.0.59' FOR SELECT FILTER BY SQL('test.test') WITH(application_name='gsq',table_num='1',estimate_row='10',is_warning='false');
(1 row)
```

Switch back to the user, connect to the database, and query the following SQL statement again. If information similar to "the statement is intercepted: ERROR: hit block rule query\_block xxx" is displayed, the statement is intercepted.

```
SELECT sum(b) FROM test.test GROUP BY a;
```

```
gaussdb=> SELECT sum(b) FROM test.test GROUP BY a;
ERROR: hit block rule query_block(user_name: user1, client_addr: 192.168.8.59, application_name: gsql, block_type: SELECT, regexp_sql: test.test, is_warning: false, table_num: 1(1), estimate_row: 10(2000000))
```

## Example: Intercept SQL Statements Based on the `unique_sql_id` or `sql_hash`

Continue with the preceding example. Switch back to the system administrator, connect to the database, and run the following command to query `unique_sql_id` and `sql_hash`:

```
EXPLAIN VERBOSE SELECT sum(b) FROM test.test GROUP BY a;
```

Record `unique_sql_id` and `sql_hash` in the command output.

```
===== Query Summary =====

System available mem: 1269760KB
Query Max mem: 1269760KB
Query estimated mem: 1919KB
Turbo Engine: true
Parser runtime: 0.071 ms
Planner runtime: 0.337 ms
Unique SQL Id: 3324222175
Unique SQL Hash: sql_170d0c75c472043b0cd35e05903c382f
(34 rows)
```

Delete the original filtering rule and create one based on `unique_sql_id`.

```
DROP BLOCK RULE query_block;
CREATE BLOCK RULE query_block FILTER BY TEMPLATE(unique_sql_id='3324222175');
```

Switch to `user1` again. An error is reported and the statement is intercepted.

```
SELECT sum(b) FROM test.test GROUP BY a;
```

```
gaussdb=> SELECT sum(b) FROM test.test GROUP BY a;
ERROR: hit block rule query_block(unique_sql_id: 3324222175)
```

Similarly, execute the statement as the system administrator. The statement is intercepted. It can be concluded that the `unique_sql_id` is used to filter and intercept SQL statements regardless of users. The SQL statement executed by all users is intercepted, and the interception effect is better.

```
SELECT sum(b) FROM test.test GROUP BY a;
```

```
gaussdb=> SELECT sum(b) FROM test.test GROUP BY a;
ERROR: hit block rule query_block(unique_sql_id: 3324222175)
```

Delete the filtering rule and create one based on `sql_hash`.

```
DROP BLOCK RULE query_block;
CREATE BLOCK RULE query_block FILTER BY
TEMPLATE(sql_hash='sql_170d0c75c472043b0cd35e05903c382f');
```

Switch to `user1` again. An error is reported and the statement is intercepted.



```
gaussdb=> SELECT sum(b) FROM test.test GROUP BY a;
ERROR: hit block rule query_block(sql_hash: sql_170d0c75c472043b0cd35e05903c382f)
```

## Helpful Links

### ALTER BLOCK RULE, DROP BLOCK RULE

This document provides basic SQL syntax. You can also configure the query filtering rules on the console. For details, see "Using DWS Query Filters to Intercept Slow SQL Statements" in the *User Guide*.

## FAQ: Why Does the Statement Fail to Be Intercepted After a Query Filtering Rule Is Created?

Some fields may not be intercepted as expected in different plans. For example:

```
CREATE BLOCK RULE test filter BY SQL('test')with(estimate_row='3');
SELECT * FROM test;
```

```
c1 | c2
---+---
 1 | 2
 1 | 2
 1 | 2
 1 | 2
 1 | 2
(5 rows)
```

In this case, the statement keyword is matched, and the number of rows queried exceeds the upper limit of three rows, but the statement is not intercepted.

```
postgres=# explain verbose select * from test;
 QUERY PLAN

id | operation | E-rows | E-distinct | E-width | E-costs
---+-----
 1 | -> Data Node Scan on "__REMOTE_FQS_QUERY__" | 0 | | 0 | 0.00

Targetlist Information (identified by plan id)

 1 --Data Node Scan on "__REMOTE_FQS_QUERY__"
 Output: test.c1, test.c2
 Node/s: All datanodes (node_group, bucket:16384)
 Remote query: SELECT c1, c2 FROM public.test
```

The **FQS (fast\_query\_shipping)** plan is used, that is, the **enable\_fast\_query\_shipping** parameter is set to **on**. As a result, no estimation information is available. The interception fails. CN lightweight plan will cause the same result. If the statement is forced to use the stream plan, the interception is successful.

```
SET enable_stream_operator=on;
SET enable_fast_query_shipping=off;
```

```
postgres=# set enable_stream_operator=on;
SET
postgres=# set enable_fast_query_shipping=off;
SET
postgres=# select * from test;
ERROR: hit block rule test(regex_sql: test, estimate_row: 3(5))
postgres=# explain verbose select * from test;
 QUERY PLAN

id | operation | E-rows | E-distinct | E-width | E-costs
---+-----+-----+-----+-----+
 1 | -> Row Adapter | 5 | | 8 | 69.00
 2 | -> Vector Streaming (type: GATHER) | 5 | | 8 | 69.00
 3 | -> CStore Scan on public.test | 5 | | 8 | 59.01

Targetlist Information (identified by plan id)

 1 --Row Adapter
 Output: c1, c2
 2 --Vector Streaming (type: GATHER)
 Output: c1, c2
 Node/s: All datanodes (node_group, bucket:16384)
 3 --CStore Scan on public.test
 Output: c1, c2
 Distribute Key: c1
```

In conclusion, if the estimation information is inaccurate, false interception or missing interception may occur. Plan information is obtained through estimation, so this error cannot be avoided.

## FAQ: How Do I Check Interception Records?

After a query filtering rule is created, many slow SQL statements are intercepted. You can view the intercepted statements in the system catalog **GS\_WLM\_SESSION\_INFO**. The **abort\_info** column records the interception information, that is, the error information of the query.

Connect to the **postgres** database and run the following command:

```
SELECT abort_info,query FROM GS_WLM_SESSION_INFO WHERE abort_info like '%hit block rule%'; --
Querying all interception records
SELECT abort_info,query FROM GS_WLM_SESSION_INFO WHERE abort_info like '%hit block rule
query_block_new%' -- Querying only the interception records whose rule name is query_block_new.
```

| query                                                                                                                                                                                                          | abort_info |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| postgres-> SELECT abort_info,query FROM OS_WLP_SESSION_TEMP WHERE abort_info like '%hit block rule%';                                                                                                          |            |
| -----                                                                                                                                                                                                          |            |
| hit block rule query_block(unique_sql_id: 3891488456)                                                                                                                                                          | select     |
| * from test.test;                                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                                                    | select     |
| * from test.test;                                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                                                    | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, client_addr: 192.168.0.59, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                         | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, client_addr: 192.168.0.59, application_name: gsql, block_type: SELECT, regex_sql: test.test, test_is_warning: false, table_num: 1(1), estimate_row: 10(20000000)) | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(unique_sql_id: 3324223175)                                                                                                                                                          | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                                                    | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                                                    | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                                                    | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(unique_sql_id: 3324223175)                                                                                                                                                          | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(unique_sql_id: 3891488456)                                                                                                                                                          | SELECT     |
| * FROM test.test;                                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, application_name: gsql, block_type: SELECT, regex_sql: test, table_num: 1(1), estimate_row: 1(30))                                                                | select     |
| * from test.test;                                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                                                    | select     |
| * from test.test;                                                                                                                                                                                              |            |
| hit block rule query_block(user_name: user1, client_addr: 192.168.0.59, application_name: gsql, block_type: SELECT, regex_sql: test.test, table_num: 1(1), estimate_row: 10(20000000))                         | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| hit block rule query_block(sql_hash: sql_1780dc75c472043bdc9e59e33c382f)                                                                                                                                       | SELECT     |
| sum(b) FROM test.test GROUP BY a;                                                                                                                                                                              |            |
| OS_WLP_SESSION_TEMP                                                                                                                                                                                            |            |

### 21.13.9.8 ALTER BLOCK RULE

## Function

This function modifies the attributes of a filtering rule, including the filtering rule name, bound client name, client IP address, user, and matching mode.

During routine database operations, executing abnormal SQL statements may use too many resources, affecting database performance and stability. You can create query filtering rules to intercept such SQL statements.

## Precautions

- This syntax is supported only by clusters of version 9.1.0.100 or later.
- Only a user with the database owner permission or the **gs\_role\_block** role permission can run the **ALTER BLOCK RULE** statement. A system administrator has this permission by default.
- Common users cannot run **CREATE BLOCK RULE**. To run it, they must be authorized by the system administrator. The authorization syntax is as follows:  
GRANT gs\_role\_block TO user;

You are advised to narrow down the application scope when creating a query filtering rule to improve performance.

- The **DEFAULT** syntax is used to reset an attribute, but it applies only to optional options.

## Syntax

```
ALTER BLOCK RULE block_name
 [[TO user_name@'host'] | [TO user_name] | [TO 'host'] | [TO DEFAULT]] |
 [FOR UPDATE | SELECT | INSERT | DELETE | MERGE | DEFAULT] |
 FILTER BY
 { SQL ('text') | TEMPLATE (template_parameter = value) }
 [WITH ({ with_parameter = value }, [...])];
ALTER BLOCK RULE block_name RENAME to new_block_name;
```

## Parameter Description

Table 21-252 ALTER BLOCK RULE parameters

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| block_name | Name of the query filtering rule whose attributes need to be modified.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | A string, which must comply with the <a href="#">naming convention</a> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| TO         | <p>Specifies the object or client IP address that determines whether the filtering rule takes effect. The options are as follows:</p> <ul style="list-style-type: none"><li>• <b>TO user_name@'host'</b>: The filtering rule takes effect only when both the username and client IP address are met.</li><li>• <b>TO user_name</b>: The query filtering rule takes effect only when the username is met, regardless of the client IP address.</li><li>• <b>TO 'host'</b>: The query filtering rule takes effect only when the client IP address is met, regardless of the username.</li></ul> | <p>The query filtering rule takes effect only when the client IP address is <b>192.168.0.59</b> and the login username is <b>user1</b>.</p> <pre>ALTER BLOCK RULE query_block TO user1@'192.168.0.59' FOR SELECT FILTER BY SQL('test.test');</pre> <p>The query filtering rule takes effect only when the login username is <b>user1</b>, regardless of the client IP address.</p> <pre>ALTER BLOCK RULE query_block TO user1 FOR SELECT FILTER BY SQL('test.test');</pre> <p>The query filtering rule takes effect only when the client IP address is <b>192.168.0.59</b>, regardless of the login user.</p> <pre>ALTER BLOCK RULE query_block TO '192.168.0.59' FOR SELECT FILTER BY SQL('test.test');</pre> |

| Parameter | Description                                                                                   | Value Range |
|-----------|-----------------------------------------------------------------------------------------------|-------------|
| FOR       | Specifies the statement type. Its value can be UPDATE, SELECT, INSERT, DELETE, or MERGE INTO. | -           |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FILTER BY | <p>Specifies the filtering method.<br/>Value:</p> <ul style="list-style-type: none"><li>• <b>SQL:</b> SQL statements are filtered based on its content. Regular expression matching is performed on statements based on keywords. For example, the length of a table name cannot exceed 1024 characters. It is recommended that the keywords be as simple as possible.</li><li>• <b>TEMPLATE:</b> SQL statements are filtered based on templates. This method intercepts specific statements executed by all users, which is more efficient. The options are as follows:<ul style="list-style-type: none"><li>– <b>unique_sql_id:</b> normalized 64-bit hash value, which is more likely to be repeated than <b>sql_hash</b>.</li></ul></li></ul> | <p>Query the values of <b>unique_sql_id</b> and <b>sql_hash</b> using EXPLAIN VERBOSE.<br/>EXPLAIN VERBOSE SELECT * FROM test.test;</p> <pre>gaussdb-&gt; explain verbose select * from test.test; QUERY PLAN ----- Id            operation            E-rows   E-distinct   E-width   E-costs -- ----- 1   -&gt; Data Node Scan on "_REMOTE_FQS_QUERY_"        0                         0   0.00  Targetlist Information (identified by plan id) ----- 1 --Data Node Scan on "_REMOTE_FQS_QUERY_"    Output: test.id, test.name    Node/s: All datanodes (group_version1, bucket:16384)    Remote query: SELECT id, name FROM test.test  ===== Query Summary ===== Parser runtime: 0.035 ms Planner runtime: 0.058 ms Unique SQL Id: 3891488456 Unique SQL Hash: sql_d557bf647fa6d1e63fbc0eb7fe6fac87 (17 rows)</pre> |

| Parameter      | Description                                                                                                                                                                      | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | <ul style="list-style-type: none"><li>– <b>sql_hash</b>: normalized hash value (MD5), which is generally unique and is recommended compared with <b>unique_sql_id</b>.</li></ul> |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| with_parameter | Parameters for querying filter rule details.                                                                                                                                     | <ul style="list-style-type: none"><li>• <b>application_name</b>: client name.</li><li>• <b>query_band</b>: load identifier.</li><li>• <b>table_num</b>: number of base tables scanned by the statement.</li><li>• <b>partition_num</b>: maximum number of partitions to be scanned by the operator.</li><li>• <b>estimate_row</b>: estimated maximum number of table rows scanned by the operator.</li><li>• <b>resource_pool</b>: name of the resource pool to be switched to.</li><li>• <b>max_active_num</b>: maximum number of concurrent statements corresponding to the rule.</li><li>• <b>is_warning</b>: whether an alarm or error is reported when a statement is intercepted. <b>on</b> indicates that an alarm is reported, and <b>off</b> (default value) indicates that an error is reported.</li></ul> |

## Examples

Create a query rule **query\_block**.

```
CREATE BLOCK RULE query_block FILTER BY SQL('update table_table set a=1');
```

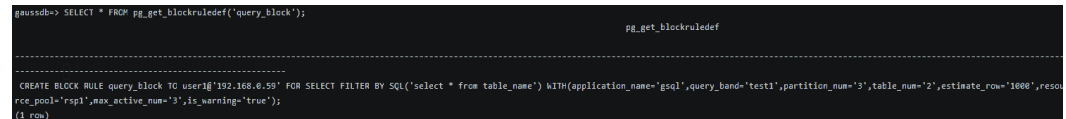
Modify the attributes of the query rule **query\_block**.

```
ALTER BLOCK RULE query_block
TO user1@'192.168.0.59'
FOR SELECT
FILTER BY
SQL('select * from table_name')
```

```
WITH (application_name='gsq!',
query_band='test1',
table_num='2',
partition_num='3',
estimate_row='1000',
resource_pool='rsp1',
max_active_num='3',
is_warning='off'
);
```

Query the modified filter rule through the **pg\_get\_blockruledef** view.

```
SELECT * FROM pg_get_blockruledef('query_block');
```



```
gaussdb> SELECT * FROM pg_get_blockruledef('query_block');
pg_get_blockruledef

CREATE BLOCK RULE query_block TO user1@'192.168.0.59' FOR SELECT FILTER BY SQL('select * from table_name') WITH(application_name='gsq!',query_band='test1',partition_num='3',table_num='2',estimate_row='1000',resource_pool='rsp1',max_active_num='3',is_warning='true');
(1 row)
```

Rename the query rule **query\_block**.

```
ALTER BLOCK RULE query_block RENAME TO query_block_new;
```

Note that if only some parameters are reset when the rule is modified, the original values of other parameters are retained. In other words, only the specified parameters are modified, and all parameters are not overwritten or cleared.

```
ALTER BLOCK RULE query_block_new
TO user1@'192.168.0.59'
FOR SELECT
FILTER BY
SQL('select * from table_name')
WITH (
query_band='test2',
table_num='3');
SELECT * FROM pg_get_blockruledef('query_block_new');
```

The command output shows that only the values of **query\_band** and **table\_num** are changed, and the values of other parameters are retained.



```
gaussdb> SELECT * FROM pg_get_blockruledef('query_block_new');
pg_get_blockruledef

CREATE BLOCK RULE query_block_new TO user1@'192.168.0.59' FOR SELECT FILTER BY SQL('select * from table_name') WITH(application_name='gsq!',query_band='test2',partition_num='3',table_num='3',estimate_row='1000',resource_pool='rsp1',max_active_num='3',is_warning='true');
(1 row)
```

Similarly, new filter criteria will not clear the previous filter criteria. For example:

```
CREATE BLOCK RULE query_block_1 FILTER BY SQL('update table_table set a=1')
WITH (application_name='gsq!',
query_band='test1');

ALTER BLOCK RULE query_block_1 FILTER BY SQL('update table_table set a=1')
WITH (
table_num='2',
partition_num='3',
estimate_row='1000',
resource_pool='rsp1',
max_active_num='3',
is_warning='off'
);
SELECT * FROM pg_get_blockruledef('query_block');
```

The command output shows that the original **application\_name** and **query\_band** parameters are retained.



```
gusssdb-> SELECT * FROM pg_get_blockruledef('query_block');
pg_get_blockruledef

CREATE BLOCK RULE query_block TO user3@192.168.0.59 FOR SELECT FILTER BY SQL('select * from table_name') WITH(application_name='gsol',query_bands='test1',table_num='2',estimate_row='1000',resource_pool='rsp1',as
x_active_num='3',is_warning='true');
(1 row)
```

Helpful Links

[CREATE BLOCK RULE, DROP BLOCK RULE](#)

21.13.9.9 DROP BLOCK RULE

Function

DROP BLOCK RULE deletes a query filtering rule. A query filtering rule includes the filtering rule name, bound client name, client IP address, user, and matching mode. During routine database operations, executing abnormal SQL statements may use too many resources, affecting database performance and stability. You can create query filtering rules to intercept such SQL statements.

Precautions

- This syntax is supported only by clusters of 9.1.0.100 and later versions.
- Only a user with the database owner permission or the **gs\_role\_block** role permission can run the **DROP BLOCK RULE** command. The system administrator has this permission by default.
- Common users cannot run **CREATE BLOCK RULE**. To run it, they must be authorized by the system administrator. The authorization syntax is as follows:  
GRANT gs\_role\_block TO user;  
You are advised to narrow down the application scope when querying filtering rules to improve performance.

Syntax

```
DROP BLOCK RULE [IF EXISTS] block_name;
```

Parameter Description

Table 21-253 DROP BLOCK RULE parameters

| Parameter  | Description                                                                               | Value Range                                     |
|------------|-------------------------------------------------------------------------------------------|-------------------------------------------------|
| IF EXISTS  | Sends a message instead of an error if the specified query filtering rule does not exist. | -                                               |
| block_name | Name of the query filtering rule to be deleted.                                           | The value must be an existing filter rule name. |

Examples

Create a query filtering rule named **query\_block**.

```
CREATE BLOCK RULE query_block FILTER BY SQL('update table_table set a=1');
```

Delete a query filtering rule named **query\_block**.

```
DROP BLOCK RULE query_block;
DROP BLOCK RULE IF EXISTS query_block;
```

## Helpful Links

[CREATE BLOCK RULE](#), [ALTER BLOCK RULE](#)

## 21.13.10 Performance and Execution Plans

### 21.13.10.1 CREATE STATISTICS

#### Function

Creates an extended statistics object for a table in the current database. The object is owned by the user who executes the command. This syntax is supported only by clusters of version 8.2.1.200 or later (not supported by clusters of version 8.3.0).

#### Precautions

- Currently, this command can only be used to create expression statistics.
- Expression statistics cannot be created for temporary tables.
- Expression statistics cannot be created on partitions of a table.
- Statistics cannot be created for expressions that contain aggregate functions, subqueries, window functions, and return sets.

#### Syntax

```
CREATE STATISTICS [IF NOT EXISTS] statistics_name
ON (expression) [, ...]
FROM table_name
```

#### Parameter Description

**Table 21-254** CREATE STATISTICS parameters

| Parameter       | Description                                                                                                                                          | Value Range                                                                          |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| statistics_name | Indicates the name of the new statistics object.                                                                                                     | A string, which must comply with the <a href="#">identifier naming conventions</a> . |
| expression      | Indicates the expression on which statistics are collected. One or more expressions are supported. Multiple expressions are separated by commas (,). | -                                                                                    |

| Parameter  | Description                                                           | Value Range |
|------------|-----------------------------------------------------------------------|-------------|
| table_name | Indicates the name of the table whose statistics are to be collected. | -           |

## Examples

Creates a sample table and insert data into it.

```
CREATE TABLE t1_row(id int, dt timestamp) WITH (orientation = row) DISTRIBUTE BY HASH (id);
INSERT INTO t1_row values (generate_series(1, 721), generate_series('2023-01-01'::timestamp,
'2023-01-31'::timestamp, '1 hour'::interval));
```

Create an expression statistics object for the table and run the ANALYZE command to collect statistics.

```
CREATE STATISTICS s1_t1_row ON EXTRACT(HOUR FROM dt), EXTRACT(DAY FROM dt) FROM t1_row;
ANALYZE t1_row;
```

View the query plan of the statement.

```
SET enable_fast_query_shipping = OFF;
EXPLAIN VERBOSE SELECT * FROM t1_row WHERE EXTRACT(HOUR FROM dt) BETWEEN 0 AND 12;
```

## Helpful Links

[DROP STATISTICS](#)

### 21.13.10.2 DROP STATISTICS

#### Function

Deletes an extended statistics object. This syntax is supported only by clusters of version 8.2.1.200 or later (not supported by clusters of version 8.3.0).

#### Precautions

None

#### Syntax

```
DROP STATISTICS [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

#### Parameter Description

**Table 21-255** DROP STATISTICS parameters

| Parameter | Description                                                                                                     | Value Range |
|-----------|-----------------------------------------------------------------------------------------------------------------|-------------|
| IF EXISTS | Sends a notification rather than reporting an error if the specified extended statistics object does not exist. | -           |

| Parameter          | Description                                                                            | Value Range                                  |
|--------------------|----------------------------------------------------------------------------------------|----------------------------------------------|
| name               | Indicates the name of the extended statistics object to be deleted.                    | An existing extended statistics object name. |
| CASCADE   RESTRICT | This parameter does not take effect. No objects depend on extended statistics objects. | -                                            |

## Examples

Deletes an extended statistics object.  
`DROP STATISTICS s1_t1_row;`

## Helpful Links

[CREATE STATISTICS](#)

## 21.13.11 Connections, Sessions, and Resource Clearing

### 21.13.11.1 CLEAN CONNECTION

#### Function

**CLEAN CONNECTION** clears database connections when a database is abnormal. You may use this statement to delete a specific user's connections to a specified database.

#### Precautions

You must specify either **dbname** or **username**.

#### Syntax

```
CLEAN CONNECTION
TO { COORDINATOR (nodename [, ...]) | NODE (nodename [, ...]) } ALL [CHECK] [FORCE] }
[FOR DATABASE dbname]
[TO USER username];
```

## Parameter Description

**Table 21-256** CLEAN CONNECTION parameters

| Parameter                                                                     | Description                                                                                                                                                                                                                                                                                                                                          | Value Range                               |
|-------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| CHECK                                                                         | This parameter can be specified only when the node list is specified as <b>TO ALL</b> . Setting this parameter will check whether a database is accessed by other sessions before its connections are cleared. If any sessions are detected before <b>DROP DATABASE</b> is executed, an error will be reported and the database will not be deleted. | -                                         |
| FORCE                                                                         | This parameter can be specified only when the node list is specified as <b>TO ALL</b> . Setting this parameter will send SIGTERM signals to all the threads related to the specified <b>dbname</b> and <b>username</b> and forcibly shut them down.                                                                                                  | -                                         |
| COORDINATOR<br>( nodename [, ... ] )  <br>NODE ( nodename<br>[, ... ] )   ALL | Deletes connections on a specified node. There are three scenarios: <ul style="list-style-type: none"><li>• Deletes connections to a specified CN.</li><li>• Deletes connections to a specified DN.</li><li>• Deletes connections to all CNs and DNs.</li></ul>                                                                                      | <b>nodename</b> is an existing node name. |
| dbname                                                                        | Deletes connections to a specific database. If this parameter is not specified, connections to all databases will be deleted.                                                                                                                                                                                                                        | Valid database name.                      |
| username                                                                      | Deletes connections of a specific user. If this parameter is not specified, connections of all users will be deleted.                                                                                                                                                                                                                                | Valid username.                           |

### Examples

Clean connections to nodes dn1 and dn2 for the **template1** database.

```
CLEAN CONNECTION TO NODE (dn1,dn2) FOR DATABASE template1;
```

Clean user **jack**'s connections to dn1.

```
CLEAN CONNECTION TO NODE (dn1) TO USER jack;
```

Delete all connections to the gaussdb database.

```
CLEAN CONNECTION TO ALL FORCE FOR DATABASE gaussdb;
```

### 21.13.11.2 ALTER SYSTEM KILL SESSION

#### Function

**ALTER SYSTEM KILL SESSION** ends a session.

#### Precautions

None

#### Syntax

```
ALTER SYSTEM KILL SESSION 'session_sid, serial' [IMMEDIATE];
```

#### Parameter Description

**Table 21-257** ALTER SYSTEM KILL SESSION parameters

| Parameter           | Description                                                                     | Value Range                                                                                                        |
|---------------------|---------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| session_sid, serial | Specifies the SID and SERIAL of a session.                                      | The <b>SIDs</b> and <b>SERIALs</b> of all sessions that can be queried from the system catalog <b>V\$SESSION</b> . |
| IMMEDIATE           | Indicates that a session will be ended instantly after the command is executed. | -                                                                                                                  |

### Examples

Query session information.

```
SELECT sid,serial#,username FROM V$SESSION;
```

| sid             | serial# | username |
|-----------------|---------|----------|
| 140131075880720 | 0       |          |
| 140131025549072 | 0       |          |
| 140131073779472 | 0       |          |
| 140131071678224 | 0       |          |
| 140131125774096 | 0       |          |

```
140131127875344 | 0 |
140131113629456 | 0 |
140131094742800 | 0 |
(8 rows)
```

End the session whose SID is 140131075880720.

```
ALTER SYSTEM KILL SESSION '140131075880720,0' IMMEDIATE;
```

### 21.13.11.3 ALTER SESSION

#### Function

**ALTER SESSION** defines or modifies the conditions or parameters that affect the current session. Modified session parameters are kept until the current session is disconnected.

#### Precautions

- If the **START TRANSACTION** command is not executed before the **SET TRANSACTION** command, the transaction is ended instantly and the command does not take effect.
- You can use the transaction\_mode(s) method declared in the **START TRANSACTION** command to avoid using the **SET TRANSACTION** command.

#### Syntax

- Set transaction parameters of a session.  

```
ALTER SESSION SET [SESSION CHARACTERISTICS AS] TRANSACTION
 { ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED } | { READ ONLY | READ
 WRITE } } [, ...] ;
```
- Set other running parameters of a session.  

```
ALTER SESSION SET
 {{config_parameter { { TO | = } { value | DEFAULT }
 | FROM CURRENT }} | CURRENT_SCHEMA [TO | =] { schema | DEFAULT }
 | TIME_ZONE time_zone
 | SCHEMA schema
 | NAMES encoding_name
 | ROLE role_name PASSWORD 'password'
 | SESSION AUTHORIZATION { role_name PASSWORD 'password' | DEFAULT }
 | XML OPTION { DOCUMENT | CONTENT }
 } ;
```

## Parameter Description

**Table 21-258** ALTER SESSION parameters

| Parameter             | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                           |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SESSION               | Indicates that the specified parameters take effect for the current session. This is the default value if neither <b>SESSION</b> nor <b>LOCAL</b> appears.<br><br>If <b>SET</b> or <b>SET SESSION</b> is executed within a transaction that is later aborted, the effects of the <b>SET</b> command disappear when the transaction is rolled back. Once the surrounding transaction is committed, the effects will persist until the end of the session, unless overridden by another <b>SET</b> . | -                                                                                                                                                                                                     |
| config_parameter      | Indicates the configurable run-time parameters. You can use <b>SHOW ALL</b> to view available run-time parameters.<br><br><b>NOTE</b><br>Some parameters that viewed by <b>SHOW ALL</b> cannot be set by <b>SET</b> . For example, <b>max_datanodes</b> .                                                                                                                                                                                                                                          | -                                                                                                                                                                                                     |
| value                 | Indicates the new value of the <b>config_parameter</b> parameter.                                                                                                                                                                                                                                                                                                                                                                                                                                  | This parameter can be specified as string constants, identifiers, numbers, or comma-separated lists of these. <b>DEFAULT</b> can be written to indicate resetting the parameter to its default value. |
| TIME_ZONE<br>timezone | Indicates the local time zone for the current session.                                                                                                                                                                                                                                                                                                                                                                                                                                             | A valid local time zone. The corresponding run-time parameter is <b>TimeZone</b> . The default value is <b>PRC</b> .                                                                                  |
| CURRENT_SCHEMA<br>A   | Indicates the current schema.                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Name of an existing schema.                                                                                                                                                                           |
| SCHEMA schema         | Indicates the current schema. Here the schema is a string.<br><br>Example: set schema 'public';                                                                                                                                                                                                                                                                                                                                                                                                    | -                                                                                                                                                                                                     |



| Parameter              | Description                                                                                                               | Value Range                                                                                                                                            |
|------------------------|---------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| NAMES<br>encoding_name | Indicates the client character encoding name. This command is equivalent to <b>set client_encoding to encoding_name</b> . | A valid character encoding name. The run-time parameter corresponding to this option is <b>client_encoding</b> . The default encoding is <b>UTF8</b> . |
| XML OPTION<br>option   | Indicates the XML resolution mode.                                                                                        | <ul style="list-style-type: none"><li>• CONTENT (default)</li><li>• DOCUMENT</li></ul>                                                                 |

## Examples

Create the **ds** schema.

```
CREATE SCHEMA ds;
```

Set the search path of the schema.

```
SET SEARCH_PATH TO ds, public;
```

Set the time/date type to the traditional postgres format (date before month).

```
SET DATESTYLE TO postgres, dmy;
```

Set the character code of the current session to UTF8.

```
ALTER SESSION SET NAMES 'UTF8';
```

Set the time zone to Berkeley of California.

```
SET TIME ZONE 'PST8PDT';
```

Set the time zone to Italy.

```
SET TIME ZONE 'Europe/Rome';
```

Set the current schema.

```
ALTER SESSION SET CURRENT_SCHEMA TO tpceds;
```

Set **XML OPTION** to **DOCUMENT**.

```
ALTER SESSION SET XML OPTION DOCUMENT;
```

Create the role **joe**, and set the session role to **joe**.

```
CREATE ROLE joe WITH PASSWORD '{password}';
ALTER SESSION SET SESSION AUTHORIZATION joe PASSWORD '{password}';
```

Switch to the default user.

```
ALTER SESSION SET SESSION AUTHORIZATION default;
```

## Helpful Links

[SET](#)

## 21.13.11.4 DISCARD

### Function

Releases internal resources related to database sessions.

The **DISCARD** command is used to reset the status of some or all sessions. Different **DISCARD** clauses release different types of resources. The **DISCARD ALL** command releases all temporary resources related to the current session and resets them to the initial state.

### Precautions

- After the **DISCARD VOLATILE { TEMPORARY | TEMP }** statement is executed, all volatile temporary table resources in the current session will be cleared. However, the statement cannot clear a single volatile temporary table resource.
- If a global temporary table occupies resources in a session, you need to run the **DISCARD** command to clear the resources of all sessions before performing DDL operations.
- After **DISCARD ALL** is executed successfully, schemas starting with **pg\_temp** and **pg\_toast\_temp** are also deleted.
- **DISCARD ALL** cannot be executed in a transaction.

### Syntax

```
DISCARD {{ GLOBAL { TEMPORARY | TEMP } [TABLE table_name] } |
 { VOLATILE { TEMPORARY | TEMP } } |
 { ALL | TEMP | TEMPORARY | PLANS | SEQUENCES }}
```

### Parameter Description

Table 21-259 DISCARD parameters

| Parameter                                        | Description                                                                                                                                                                                                                                                                                                            |
|--------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VOLATILE { TEMPORARY   TEMP }                    | Releases resources related to the VOLATILE temporary table in the current session.                                                                                                                                                                                                                                     |
| GLOBAL { TEMPORARY   TEMP } [ TABLE table_name ] | <ul style="list-style-type: none"><li>• Run the <b>DISCARD GLOBAL TEMP</b> command to release resources related to the global temporary table in the current session.</li><li>• <b>DISCARD GLOBAL TEMP TABLE table_name</b> releases resources of a specified global temporary table in the current session.</li></ul> |
| TEMP   TEMPORARY                                 | Releases resources related to all temporary tables in the current session, including volatile and global temporary tables.                                                                                                                                                                                             |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                          |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PLANS     | Releases all cached query plans in the current session and forces them to be replanned when related <b>PREPARE</b> statements are used next time.                                                                                                                                                                                                                                    |
| SEQUENCES | Discards all cached sequence-related states, including <b>curval()/lastval()</b> information and any pre-allocated sequence values that have not been returned through <b>nextval()</b> .                                                                                                                                                                                            |
| ALL       | Releases all temporary resources related to the current session and resets them to their initial state. This has almost the same effect as executing the following statement sequence:<br>SET SESSION AUTHORIZATION DEFAULT;<br>RESET ALL;<br>DEALLOCATE ALL;<br>CLOSE ALL;<br>UNLISTEN *;<br>SELECT pg_advisory_unlock_all();<br>DISCARD PLANS; DISCARD SEQUENCES;<br>DISCARD TEMP; |

## Examples

### DISCARD global temporary tables

```
CREATE GLOBAL TEMP TABLE t_global_temp(a int,b int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

INSERT INTO t_global_temp VALUES(1,1),(2,2);
INSERT 0 2

DROP TABLE t_global_temp;
ERROR: can not DROP TABLE when global temp table "t_global_temp" is in use

DISCARD GLOBAL TEMP TABLE t_global_temp;

DROP TABLE t_global_temp;
DROP TABLE
```

The **DISCARD VOLATILE** command clears all resources related to volatile temporary tables in the current session.

```
CREATE VOLATILE TEMP TABLE TX1(A INT) DISTRIBUTE BY HASH(A);
CREATE TABLE
CREATE VOLATILE TEMP TABLE TX2(A INT) DISTRIBUTE BY HASH(A);
CREATE TABLE

SELECT * FROM TX1;
a

(0 rows)
SELECT * FROM TX2;
a

(0 rows)

DISCARD VOLATILE TEMP;
```

```
SELECT * FROM TX1;
ERROR: relation "tx1" does not exist
LINE 1: SELECT * FROM TX1;
 ^
SELECT * FROM TX2;
ERROR: relation "tx2" does not exist
LINE 1: SELECT * FROM TX2;
```

After **DISCARD TEMP** is run, all temporary table resources in the current session are cleared.

```
CREATE GLOBAL TEMP TABLE t_global_temp(a int,b int);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using round-robin as the distribution mode by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE
INSERT INTO t_global_temp VALUES(1,1),(2,2);
INSERT 0 2

CREATE VOLATILE TEMP TABLE t_volatile_temp(a int,b int);
CREATE TEMP TABLE t_temp(a int,b int);

DISCARD TEMP;

SELECT * FROM t_global_temp;
a | b
---+---
(0 rows)

SELECT * FROM t_volatile_temp;
ERROR: relation "t_volatile_temp" does not exist
LINE 1: select * from t_volatile_temp;

SELECT * FROM t_temp;
ERROR: relation "t_temp" does not exist
LINE 1: select * from t_temp;
```

## 21.13.11.5 TRUNCATE

### Function

**TRUNCATE** quickly removes all rows from a database table.

It has the same effect as an unqualified DELETE, but since it does not actually scan the table, it is faster. This is most effective on large tables.

## Differences Among TRUNCATE TABLE, DELETE TABLE, and DROP TABLE

**Table 21-260** Differences among TRUNCATE TABLE, DELETE TABLE, and DROP TABLE

| Dimension       | TRUNCATE TABLE                                                         | DELETE TABLE                                                   | DROP TABLE                          |
|-----------------|------------------------------------------------------------------------|----------------------------------------------------------------|-------------------------------------|
| Syntax          | DDL                                                                    | DML                                                            | DDL                                 |
| Deleted content | All data in the table is deleted, but the table schema is not deleted. | Only the content is deleted and the definition is not deleted. | Content and definition are deleted. |
| Space release   | The space is released.                                                 | The space is not released.                                     | The space is released.              |

| Dimension       | TRUNCATE TABLE                                                                                                                                                                              | DELETE TABLE                                                                                           | DROP TABLE                                                                |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| Execution Speed | Fast<br>Data is deleted by releasing the data pages used to store table data. Only the page release is recorded in the transaction log. Less system and transaction log resources are used. | Slow<br>Each time a row is deleted, a record is generated for each deleted row in the transaction log. | Fastest                                                                   |
| Scenario        | You need to quickly delete data from a table while retaining the table schema, and the data volume is large.                                                                                | You need to delete data based on specific conditions and the data volume is under control.             | You need to delete the entire table, including the table schema and data. |

## Precautions

- Exercise caution when running the **TRUNCATE TABLE** statement. Before running this statement, ensure that the table data can be deleted or has been backed up. After you run the **TRUNCATE TABLE** statement to delete table data, the data cannot be restored.
- The **TRUNCATE** operation on global temporary tables only truncates data of the current session. Data of other sessions is not affected.
- In the storage-compute decoupling architecture, it is not possible to perform the **TRUNCATE** operation on common tables and temporary tables simultaneously.



### WARNING

- Avoid performing **ALTER TABLE**, **ALTER TABLE PARTITION**, **DROP PARTITION**, and **TRUNCATE** operations during peak hours to prevent long SQL statements from blocking these operations or SQL services.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

## Syntax

**TRUNCATE** empties a table or set of tables.

```
TRUNCATE [TABLE] [ONLY] { [[database_name.]schema_name.]table_name [*] } [, ...]
[CONTINUE IDENTITY] [CASCADE | RESTRICT] ;
```

## Parameter Description

**Table 21-261** TRUNCATE parameters

| Parameter          | Description                                                                                                                                                   | Value Range                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ONLY               | Specifies the range of tables to be cleared.<br><br>This parameter is reserved only for compatibility with PostgreSQL. DWS does not support inherited tables. | <ul style="list-style-type: none"><li>If <b>ONLY</b> is specified, only the specified table is cleared.</li><li>If <b>ONLY</b> is not specified, the specified table and all its inherited tables (if any) are cleared.</li></ul>                                                                                                                                                                       |
| database_name      | Specifies the name of the database where the table to be cleared is located.                                                                                  | An existing database name.                                                                                                                                                                                                                                                                                                                                                                              |
| schema_name        | Specifies the schema name of the table to be cleared.                                                                                                         | An existing schema name.                                                                                                                                                                                                                                                                                                                                                                                |
| table_name         | Specifies the name of the table to be cleared (which can be schema-qualified).                                                                                | An existing table name.                                                                                                                                                                                                                                                                                                                                                                                 |
| CONTINUE IDENTITY  | Does not change the values of sequences.                                                                                                                      | This parameter is set to the default value.                                                                                                                                                                                                                                                                                                                                                             |
| CASCADE   RESTRICT | (Optional) Specifies the method of clearing tables that have foreign key references.                                                                          | <ul style="list-style-type: none"><li><b>CASCADE</b>: automatically truncates all tables that have foreign-key references to any of the named tables, or to any tables added to the group due to <b>CASCADE</b>.</li><li><b>RESTRICT</b>: refuses to truncate if any of the tables have foreign-key references from tables that are not listed in the command.</li></ul> The default value is RESTRICT. |

## Examples

1. Creates the sample table **test\_t** and insert data into it.

```
CREATE TABLE test_t
(
 col_id INT PRIMARY KEY,
 col_number INT NOT NULL,
 col_date DATE NOT NULL,
 col_price NUMERIC (10,2),
 col_status TEXT
)
```

```
WITH (ORIENTATION = COLUMN)
DISTRIBUTE BY HASH (col_id);

INSERT INTO test_t VALUES
(01,300,'2025-03-01',99.20,'sold'),
(02,400,'2025-04-01',95.50,'sold'),
(03,450,'2025-05-01',420.50,'sold'),
(04,100,'2025-06-01',100.85,'restock');
```

2. View data in the **test\_t** table.

```
SELECT * FROM test_t;
```

| col_id | col_number | col_date            | col_price | col_status |
|--------|------------|---------------------|-----------|------------|
| 1      | 300        | 2025-03-01 00:00:00 | 99.20     | sold       |
| 2      | 400        | 2025-04-01 00:00:00 | 95.50     | sold       |
| 3      | 450        | 2025-05-01 00:00:00 | 420.50    | sold       |
| 4      | 100        | 2025-06-01 00:00:00 | 100.85    | restock    |

(4 rows)

3. Clear the **test\_t** table.

```
TRUNCATE test_t;
```

If the following information is displayed, the table is cleared successfully.

```
TRUNCATE TABLE
```

4. View the table definition. The TRUNCATE TABLE operation clears the table but retains the table definition.

```
SELECT * FROM pg_get_tabledef('public.test_t');
pg_get_tabledef

SET search_path = public;
CREATE TABLE test_t (
 col_id integer NOT NULL,
 col_number integer NOT NULL,
 col_date timestamp(0) without time zone NOT NULL,
 col_price numeric(10,2),
 col_status text
)
WITH (orientation=column, compression=middle, colversion=2.0, enable_delta=false)+
DISTRIBUTE BY HASH(col_id)
;
(1 row)
```

## 21.13.11.6 VACUUM

### Function

**VACUUM** reclaims storage space occupied by tables or B-tree indexes. In normal database operation, rows that have been deleted are not physically removed from their table; they remain present until a **VACUUM** is done. Therefore, it is necessary to execute **VACUUM** periodically, especially on frequently-updated tables.

### Differences between VACUUM and VACUUM FULL

The **VACUUM** command comes in two types: **VACUUM** and **VACUUM FULL**. **VACUUM** performs a simpler process known as **LAZY VACUUM** compared to **VACUUM FULL**. The table below highlights the key differences between **LAZY VACUUM** and **VACUUM FULL**.

**Table 21-262** Differences between VACUUM and VACUUM FULL

| Item                 | VACUUM                                                                                                                                                                       | VACUUM FULL                                                                                                                                                                     |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Space cleanup        | When you delete a record at the end of the table, its space is freed and given back to the operating system. For records elsewhere, their space becomes available for reuse. | Regardless of location, all deleted data eventually frees up physical space for the OS. New data insertion requires allocating fresh disk pages.                                |
| Lock type            | A Level-4 lock (ShareUpdateExclusive-Lock) allows concurrent operations.                                                                                                     | A Level-8 lock (AccessExclusiveLock) locks the table entirely, pausing all other actions during its execution.                                                                  |
| Physical space       | Not released.                                                                                                                                                                | Released.                                                                                                                                                                       |
| Transaction ID       | Not reclaimed.                                                                                                                                                               | Reclaimed.                                                                                                                                                                      |
| Resource consumption | The execution requires little resources and can be performed periodically.                                                                                                   | The execution requires significant resources. Make sure the database's disk usage nears its limit before starting. Also, carry out this task when there is less data to handle. |
| Effect               | Table performance will be slightly improved.                                                                                                                                 | Table performance is greatly improved.                                                                                                                                          |

## Auto Cleanup (AUTOVACUUM)

The **VACUUM** operation frees up space, enhancing query performance. DWS offers automatic cleanup, which you can activate under specific conditions as needed. You can configure this on the console by referring to "Managing O&M Plans" in the *Data Warehouse Service (DWS) User Guide*.

## Precautions

- With no table specified, **VACUUM** processes all the tables that the current user has permission to vacuum in the current database. With a table specified, **VACUUM** processes only that table.
- To perform **VACUUM** on a table, you must be the owner of the table, granted with the **VACUUM** permission on the table, or granted with the **gs\_role\_vacuum\_any** role. By default, a system administrator has this permission. However, database owners are allowed to **VACUUM** all tables in their databases, except shared catalogs. (The restriction for shared catalogs means that a true database-wide **VACUUM** can only be executed by the system administrator). **VACUUM** skips over any tables that the calling user does not have the permission to vacuum.



- **VACUUM** cannot be executed inside a transaction block.
- It is recommended that active production databases be vacuumed frequently (at least nightly), in order to remove dead rows. **VACUUM ANALYZE** executes a **VACUUM** operation and then an **ANALYZE** operation for each selected table. After adding or deleting a large number of rows, it might be a good idea to execute the **VACUUM ANALYZE** command for the affected table. This will update the system catalogs with the results of all recent changes, and allow the optimizer to make better choices in planning queries.
- **VACUUM** causes a substantial increase in I/O traffic, which might cause poor performance for other active sessions. Therefore, it is sometimes advisable to use the cost-based **VACUUM** delay feature.
- When **VERBOSE** is specified, **VACUUM** prints progress messages to indicate which table is currently being processed. Various statistics about the tables are printed as well.
- When the option list is surrounded by parentheses, the options can be written in any order. If there are no brackets, the options must be given in the order displayed in the syntax.
- During **VACUUM** on column-store tables, four internal operations occur: **VACUUM** on the primary table, **VACUUM** on the Desc table of the primary table, **VACUUM** on the delta table of the primary table, and migration of data from the delta table to the primary table. **VACUUM** does not reclaim the storage space of the delta table. To reclaim it, do **VACUUM DELTAMERGE** to the column-store table. By default, the **VACUUM** primary table operation is enabled. You can disable this function by setting the **colvacuum\_threshold\_scale\_factor** parameter.
- The **VACUUM** operation on column-store primary tables does not support temporary tables, multi-temperature tables, and time series tables.
- This operation does not immediately free up space. To reclaim space right away, execute the `vac_fileclear_relation` function after running **VACUUM**; this will place an exclusive lock on the chosen table.
- Plain **VACUUM** (without **FULL**) recycles space and makes it available for reuse. This form of the command can operate in parallel with normal reading and writing of the table, as an exclusive lock is not obtained. **VACUUM FULL** executes wider processing, including moving rows across blocks to compress tables so they occupy minimum number of disk blocks. This form is much slower and requires an exclusive lock on each table while it is being processed.
- **FULL** is recommended only in special scenarios. For example, you wish to physically narrow the table to decrease the occupied disk space after deleting most rows of a table. **VACUUM FULL** usually shrinks more table size than **VACUUM**. If the physical space usage does not decrease after you run the command, check whether there are other active transactions (that have started before you delete data transactions and not ended before you run **VACUUM FULL**). If there are such transactions, run this command again when the transactions quit.
- **VACUUM** and **VACUUM FULL** clear deleted tuples after the delay specified by **vacuum\_defer\_cleanup\_age**.
- Running **VACUUM FULL** on system catalogs can only be done when the database is offline. Otherwise, table locks, exceptions, and errors may occur.

- Running **VACUUM FULL** on a column-store partitioned table locks the table and its partitions.
- A distributed deadlock may occur when **VACUUM FULL** and DML statements run concurrently in the following scenarios:
  - **VACUUM FULL** on subpartitions and **INSERT/UPDATE/DELETE** on the primary table
  - **VACUUM FULL** on a full table and **SELECT** on a full table or **SELECT** on subpartitions
- In the storage-compute decoupled architecture, a message is displayed indicating that the full database **VACUUM**, **VACUUM FULL**, **VACUUM ANALYZE**, or **VACUUM DELTAMERGE** is not supported.
- When you run the **VACUUM** command on a column-store 3.0 table during redistribution in the architecture with decoupled storage and compute, the system keeps the data intact and shows the message below instead of clearing it.

When you run **VACUUM**, the **VACUUM** operation is skipped.

```
VACUUM tpcds.web_returns_p1;
```

```
NOTICE: skip vacuum relation "web_returns_p1" since it is in redistribution state
```

When you run **VACUUM ANALYZE**, the **VACUUM** operation is skipped, but statistics are collected.

```
VACUUM ANALYZE tpcds.web_returns_p1;
```

```
NOTICE: skip vacuum relation "web_returns_p1" since it is in redistribution state
```



#### WARNING

- When the **VACUUM FULL** operation is performed on a table, the table and index on the table are rebuilt. During the rebuilding, data is written to a new data file. After the rebuilding is complete, the original file is deleted. If the table is large, rebuilding consumes a large amount of disk space. When the disk space is insufficient, exercise caution when performing the **VACUUM FULL** operation on large tables to prevent the cluster from being read-only.
- Execute **VACUUM FULL** on tables with a high dirty page rate and small CU ratio exceeding 25% at regular intervals. Perform **VACUUM FULL** on common tables during low-traffic hours and on system tables when they are offline.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.

## Syntax

- Reclaim space and update statistics. The keyword sequence must be given in the order in which the syntax is displayed.

```
VACUUM [({ FULL | FREEZE | VERBOSE | {ANALYZE | ANALYSE } } [...])]
[table_name [(column_name [, ...])]] [PARTITION (partition_name)];
```

- Reclaim space, without updating statistics information.

```
VACUUM [FULL [COMPACT]] [FREEZE] [VERBOSE] [table_name] [PARTITION
(partition_name)];
```

- Reclaim space and update statistics information, with a specific order of keywords required.

```
VACUUM [FULL] [FREEZE] [VERBOSE] { ANALYZE | ANALYSE } [VERBOSE]
[table_name [(column_name [, ...])]] [PARTITION (partition_name)];
```

- For HDFS and column-store tables, migrate data from the delta table to the primary table. The **partition\_name** parameter is supported only by 8.2.1.200 and later cluster versions.

```
VACUUM DELTAMERGE [table_name][partition_name];
```

- For HDFS tables, delete the empty value partition directory of HDFS table in HDFS storage.

```
VACUUM HDFSDIRECTORY [table_name];
```

## Parameter Description

Table 21-263 VACUUM parameters

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Value Range |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| FULL      | <p>Selects the parameter, which can reclaim more space, but takes much longer and exclusively locks the table.</p> <p><b>FULL</b> options can also contain the <b>COMPACT</b> parameter, which is only used for the HDFS table. Specifying the <b>COMPACT</b> parameter improves <b>VACUUM FULL</b> operation performance.</p> <p><b>NOTICE</b></p> <ul style="list-style-type: none"><li><b>COMPACT</b> and <b>PARTITION</b> cannot be used at the same time.</li><li>Using <b>FULL</b> will cause statistics missing. To collect statistics, add the keyword <b>ANALYZE</b> to <b>VACUUM FULL</b>.</li></ul> | -           |

| Parameter               | Description                                                                                                                                                                                                                                                                            | Value Range                                      |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| FREEZE                  | Resolves the transaction ID wraparound issue by default. Using FREEZE sets <b>vacuum_freeze_min_age</b> to 0 during VACUUM.<br><b>NOTICE</b><br>Since DWS employs 64-bit XIDs, the wraparound problem does not occur theoretically. Avoid using <b>VACUUM FREEZE</b> unless necessary. | -                                                |
| VERBOSE                 | Prints a detailed vacuum activity report for each table.                                                                                                                                                                                                                               | -                                                |
| ANALYZE<br> <br>ANALYSE | Updates statistics used by the optimizer to determine the most efficient way to execute a query.                                                                                                                                                                                       | -                                                |
| table_name              | Indicates the name (optionally schema-qualified) of a specific table to vacuum.                                                                                                                                                                                                        | Defaults are all tables in the current database. |
| column_name             | Indicates the name of a specific field to analyze.                                                                                                                                                                                                                                     | Defaults are all columns.                        |

| Parameter      | Description                                                                                                                                                                                                                                                                                                                    | Value Range                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| PARTITION      | <p>Specifies that the table to be cleared is a partitioned table. HDFS tables do not support the PARTITION parameter.</p> <p><b>NOTE</b><br/>If the <b>PARTITION</b> and <b>COMPACT</b> parameters are used at the same time, the following error message is displayed:<br/><b>COMPACT can not be used with PARTITION.</b></p> | -                            |
| partition_name | Indicates the partition name of a specific table to vacuum.                                                                                                                                                                                                                                                                    | Defaults are all partitions. |

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                 | Value Range |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| DELTA MERGE | For HDFS tables and column-store tables, migrate data in the delta table of HDFS tables or column-store tables to the primary table. If the data volume of the delta table is less than 60,000 rows, the data will not be migrated. Otherwise, the data will be migrated to HDFS, and the delta table will be cleared by <b>TRUNCATE</b> . For a column-store table, this operation always transfers all data in the delta table to the CU. | -           |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Value Range |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
|           | <p><b>NOTE</b></p> <p>The following DFX functions are provided to return the data storage in the delta table of a column-store table (for an HDFS table, it can be returned by <b>EXPLAIN ANALYZE</b>):</p> <ul style="list-style-type: none"><li>pgxc_get_delta_info(TEXT): The input parameter is a column-store table name. The delta table information on each node is collected and displayed, including the number of active tuples, table size, and maximum block ID.</li><li>get_delta_info(TEXT): The input parameter is a column-store table name. The system summarizes the results returned from pgxc_get_delta_info and returns the total number of active tuples, total table size, and maximum block ID in the delta table. When querying delta information about a temporary table, you need to specify the schema of the temporary table. Otherwise, an error is reported, indicating that the table cannot be found.</li></ul> |             |

| Parameter      | Description                                                                               | Value Range |
|----------------|-------------------------------------------------------------------------------------------|-------------|
| HDFS DIRECTORY | Deletes the empty value partition directory of HDFS table in HDFS storage for HDFS table. | -           |

## Examples

Delete all tables in the current database.

```
VACUUM;
```

Reclaim the space of partition **P2** of the **tpcds.web\_returns\_p1** table without updating statistics.

```
VACUUM FULL tpcds.web_returns_p1 PARTITION(P2);
```

Reclaim the space of the **tpcds.web\_returns\_p1** table and update statistics.

```
VACUUM FULL ANALYZE tpcds.web_returns_p1;
```

Delete all tables in the current database and collect statistics about the query optimizer.

```
VACUUM ANALYZE;
```

Delete only the **reason** table.

```
VACUUM (VERBOSE, ANALYZE) tpcds.reason;
```

Perform the **DELTAMERGE** operation on the column-store table **table\_delta**.

```
VACUUM DELTAMERGE tpcds.table_delta;
```

Perform the **DELTAMERGE** operation only on partition **p1** of the column-store table **table\_delta**.

```
VACUUM DELTAMERGE tpcds.table_delta partition(p1);
```

## 21.13.12 Settings and Comments

### 21.13.12.1 SHOW

#### Function

**SHOW** shows the current value of a run-time parameter. You can use the **SET** statement to set these parameters.

#### Precautions

Some parameters that can be viewed by **SHOW** are read-only. You can view but cannot modify their values.



## Syntax

```
SHOW
{
 configuration_parameter |
 CURRENT_SCHEMA |
 TIME_ZONE |
 TRANSACTION ISOLATION LEVEL |
 SESSION AUTHORIZATION |
 ALL
};
```

## Parameter Description

See [Parameter Description](#) in **RESET**.

## Examples

Show the value of **timezone**.

```
SHOW timezone;
```

Show the current setting of the **DateStyle** parameter.

```
SHOW DateStyle;
```

Show the current setting of all parameters.

```
SHOW ALL;
```

## Helpful Links

[SET](#), [RESET](#)

### 21.13.12.2 SET

## Function

**SET** modifies a run-time parameter.

## Precautions

Most run-time parameters can be modified by executing **SET**. Some parameters cannot be modified after a server or session starts.

## Syntax

- Set the system time zone.  

```
SET [SESSION | LOCAL] TIME_ZONE { timezone | LOCAL | DEFAULT };
```
- Set the schema of the table.  

```
SET [SESSION | LOCAL]
{CURRENT_SCHEMA { TO | = } { schema | DEFAULT }
| SCHEMA 'schema'};
```
- Set client encoding.  

```
SET [SESSION | LOCAL] NAMES encoding_name;
```
- Set XML parsing mode.  

```
SET [SESSION | LOCAL] XML OPTION { DOCUMENT | CONTENT };
```
- Set other running parameters.

```
SET [LOCAL | SESSION]
 { {config_parameter { { TO | = } { value | DEFAULT }
 | FROM CURRENT }}};
```

## Parameter Description

**Table 21-264** SET parameters

| Parameter                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Value Range                                                                                                          |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| SESSION                  | <p>Indicates that the specified parameters take effect for the current session. This is the default value if neither <b>SESSION</b> nor <b>LOCAL</b> appears.</p> <p>If <b>SET</b> or <b>SET SESSION</b> is executed within a transaction that is later aborted, the effects of the <b>SET</b> command disappear when the transaction is rolled back. Once the surrounding transaction is committed, the effects will persist until the end of the session, unless overridden by another <b>SET</b>.</p>                                                   | -                                                                                                                    |
| LOCAL                    | <p>Indicates that the specified parameters take effect for the current transaction. After <b>COMMIT</b> or <b>ROLLBACK</b>, the session-level setting takes effect again.</p> <p>The effects of <b>SET LOCAL</b> last only till the end of the current transaction, whether committed or not. A special case is <b>SET</b> followed by <b>SET LOCAL</b> within a single transaction: the <b>SET LOCAL</b> value will be seen until the end of the transaction, but afterwards (if the transaction is committed) the <b>SET</b> value will take effect.</p> | -                                                                                                                    |
| TIME_ZONE<br>timezone    | Indicates the local time zone for the current session.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | A valid local time zone. The corresponding run-time parameter is <b>TimeZone</b> . The default value is <b>PRC</b> . |
| CURRENT_SCHEMA<br>schema | Indicates the current schema.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Specifies an existing schema name.                                                                                   |
| SCHEMA<br>schema         | <p>Indicates the current schema. Here the schema is a string.</p> <p>Example: set schema 'public';</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                     | -                                                                                                                    |

| Parameter               | Description                                                                                                                                                                                                                                                             | Value Range                                                                                                                                            |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| NAMES<br>encoding_name  | Indicates the client character encoding name. This command is equivalent to <b>set client_encoding to encoding_name</b> .                                                                                                                                               | A valid character encoding name. The run-time parameter corresponding to this option is <b>client_encoding</b> . The default encoding is <b>UTF8</b> . |
| XML<br>OPTION<br>option | Indicates the XML resolution mode.                                                                                                                                                                                                                                      | CONTENT (default) and DOCUMENT                                                                                                                         |
| config_parameter        | Indicates the configurable run-time parameters. You can use <b>SHOW ALL</b> to view available run-time parameters.<br><b>NOTE</b><br>Some parameters that viewed by <b>SHOW ALL</b> cannot be set by <b>SET</b> . For example, <b>max_datanodes</b> .                   | -                                                                                                                                                      |
| value                   | Indicates the new value of the <b>config_parameter</b> parameter. This parameter can be specified as string constants, identifiers, numbers, or comma-separated lists of these. <b>DEFAULT</b> can be written to indicate resetting the parameter to its default value. | -                                                                                                                                                      |

## Examples

Configure the search path of the **tpcds** schema.

```
SET search_path TO tpcds, public;
```

Set the date style to the traditional POSTGRES style (date placed before month).

```
SET datestyle TO postgres;
```

## Helpful Links

[RESET](#), [SHOW](#)

### 21.13.12.3 RESET

#### Function

**RESET** restores run-time parameters to their default values. The default values are parameter default values compiled in the **postgresql.conf** configuration file.

**RESET** is an alternative spelling for:

**SET configuration\_parameter TO DEFAULT**

## Precautions

**RESET** and **SET** have the same transaction behavior. Their impact will be rolled back.

## Syntax

```
RESET {configuration_parameter | CURRENT_SCHEMA | TIME_ZONE | TRANSACTION_ISOLATION_LEVEL |
SESSION_AUTHORIZATION | ALL};
```

## Parameter Description

**Table 21-265** RESET parameters

| Parameter                   | Description                                          | Value Range                                                                                                                                                                                                 |
|-----------------------------|------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| configuration_parameter     | Specifies the name of a settable run-time parameter. | You can view run-time parameters by running the <b>SHOW ALL</b> command.<br><b>NOTE</b><br>Some parameters that viewed by <b>SHOW ALL</b> cannot be set by <b>SET</b> . For example, <b>max_datanodes</b> . |
| CURRENT_SCHEMA              | Specifies the current schema.                        | -                                                                                                                                                                                                           |
| TIME_ZONE                   | Specifies the time zone.                             | -                                                                                                                                                                                                           |
| TRANSACTION_ISOLATION_LEVEL | Specifies the transaction isolation level.           | -                                                                                                                                                                                                           |
| SESSION_AUTHORIZATION       | Specifies the session authorization.                 | -                                                                                                                                                                                                           |
| ALL                         | Specifies all runtime parameters.                    | -                                                                                                                                                                                                           |

## Examples

Reset **timezone** to the default value.

```
RESET timezone;
```

Set all parameters to their default values.

```
RESET ALL;
```

## Helpful Links

[SET](#), [SHOW](#)

## 21.13.12.4 SET CONSTRAINTS

### Function

**SET CONSTRAINTS** sets the behavior of constraint checking within the current transaction.

**IMMEDIATE** constraints are checked at the end of each statement. **DEFERRED** constraints are not checked until transaction commit. Each constraint has its own **IMMEDIATE** or **DEFERRED** mode.

Upon creation, a constraint is given one of three characteristics **DEFERRABLE INITIALLY DEFERRED**, **DEFERRABLE INITIALLY IMMEDIATE**, or **NOT DEFERRABLE**. The third class is always **IMMEDIATE** and is not affected by the **SET CONSTRAINTS** command. The first two classes start every transaction in specified modes, but its behaviors can be changed within a transaction by **SET CONSTRAINTS**.

**SET CONSTRAINTS** with a list of constraint names changes the mode of just those constraints (which must all be deferrable). If multiple constraints match a name, the name is affected by all of these constraints. **SET CONSTRAINTS ALL** changes the modes of all deferrable constraints.

When **SET CONSTRAINTS** changes the mode of a constraint from **DEFERRED** to **IMMEDIATE**, the new mode takes effect retroactively: any outstanding data modifications that would have been checked at the end of the transaction are instead checked during the execution of the **SET CONSTRAINTS** command. If any such constraint is violated, the **SET CONSTRAINTS** fails (and does not change the constraint mode). Therefore, **SET CONSTRAINTS** can be used to force checking of constraints to occur at a specific point in a transaction.

Only foreign key constraints are affected by this setting. Check and unique constraints are always checked immediately when a row is inserted or modified.

### Precautions

**SET CONSTRAINTS** sets the behavior of constraint checking only within the current transaction. Therefore, if you execute this command outside of a transaction block (**START TRANSACTION/COMMIT** pair), it will not appear to have any effect.

### Syntax

```
SET CONSTRAINTS { ALL | { name } [, ...] } { DEFERRED | IMMEDIATE };
```

### Parameter Description

Table 21-266 SET CONSTRAINTS parameters

| Parameter | Description                    | Value Range                                                                                  |
|-----------|--------------------------------|----------------------------------------------------------------------------------------------|
| name      | Specifies the constraint name. | An existing constraint name. which can be found in the system catalog <b>pg_constraint</b> . |

| Parameter | Description                                                          | Value Range |
|-----------|----------------------------------------------------------------------|-------------|
| ALL       | Indicates all constraints.                                           | -           |
| DEFERRED  | Indicates that constraints are not checked until transaction commit. | -           |
| IMMEDIATE | Indicates that constraints are checked at the end of each statement. | -           |

## Examples

Set that constraints are checked when a transaction is committed.

```
SET CONSTRAINTS ALL DEFERRED;
```

### 21.13.12.5 SET SESSION AUTHORIZATION

## Function

**SET SESSION AUTHORIZATION** sets the session user identifier and the current user identifier of the current SQL session to a specified user.

## Precautions

The session identifier can be changed only when the initial session user has the system administrator rights. Otherwise, the system supports the command only when the authenticated user name is specified.

## Syntax

- SET SESSION AUTHORIZATION** sets the session user identifier and the current user identifier of the current session.

```
SET [SESSION | LOCAL] SESSION AUTHORIZATION role_name PASSWORD '{password}';
```

- Reset the identifiers of the session and current users to the initially authenticated user names.

```
{SET [SESSION | LOCAL] SESSION AUTHORIZATION DEFAULT
| RESET SESSION AUTHORIZATION};
```

## Parameter Description

**Table 21-267** SET SESSION AUTHORIZATION parameters

| Parameter | Description                                                                  | Value Range |
|-----------|------------------------------------------------------------------------------|-------------|
| SESSION   | Indicates that the specified parameters take effect for the current session. | -           |

| Parameter | Description                                                                                       | Value Range           |
|-----------|---------------------------------------------------------------------------------------------------|-----------------------|
| LOCALE    | Indicates that the specified command takes effect only for the current transaction.               | -                     |
| role_name | Specifies the role name.                                                                          | An existing role name |
| password  | Specifies the password of a role. It must comply with the password convention.                    | -                     |
| DEFAULT   | Reset the identifiers of the session and current users to the initially authenticated user names. | -                     |

## Examples

Set the current user to **paul**.

```
SET SESSION AUTHORIZATION paul password '{password};
```

View the current session user and the current user.

```
SELECT SESSION_USER, CURRENT_USER;
```

Reset the current user.

```
RESET SESSION AUTHORIZATION;
```

## Helpful Links

[SET ROLE](#)

### 21.13.12.6 COMMENT

#### Function

**COMMENT** defines or changes the comment of an object.

#### Precautions

- Only one comment string is stored for each object. To modify a comment, issue a new **COMMENT** command for the same object. To remove a comment, write **NULL** at the location of the text string. Comments are automatically deleted when their objects are deleted.
- There is no security for viewing comments. Any user with database access can see all object comments. Comments on shared items like databases, roles, and tablespaces are visible to any user connected to any database in the cluster. Avoid adding sensitive security information to comments.
- For most objects, only the owner of the object can set comments. A role has no owner. So, only a system admin can use the **COMMENT ON ROLE** command for system admin roles. Roles with the **CREATE ROLE** permission

can add comments to non-admin roles. Administrators can comment on anything.

Syntax

```
COMMENT ON
{
 AGGREGATE agg_name (agg_type [, ...]) |
 CAST (source_type AS target_type) |
 COLLATION object_name |
 COLUMN { table_name.column_name | view_name.column_name } |
 CONSTRAINT constraint_name ON table_name |
 CONVERSION object_name |
 DATABASE object_name |
 DOMAIN object_name |
 EXTENSION object_name |
 FOREIGN DATA WRAPPER object_name |
 FOREIGN TABLE object_name |
 FUNCTION function_name ([{[argmode] [argname] argtype} [, ...]]) |
 INDEX object_name |
 LARGE OBJECT large_object_oid |
 OPERATOR operator_name (left_type, right_type) |
 OPERATOR CLASS object_name USING index_method |
 OPERATOR FAMILY object_name USING index_method |
 [PROCEDURAL] LANGUAGE object_name |
 ROLE object_name |
 RULE rule_name ON table_name |
 SCHEMA object_name |
 SERVER object_name |
 TABLE object_name |
 TABLESPACE object_name |
 TEXT SEARCH CONFIGURATION object_name |
 TEXT SEARCH DICTIONARY object_name |
 TEXT SEARCH PARSER object_name |
 TEXT SEARCH TEMPLATE object_name |
 TYPE object_name |
 VIEW object_name
}
IS 'text';
```

Parameter Description

Table 21-268 COMMENT parameters

| Parameter   | Description                                                       | Value Range |
|-------------|-------------------------------------------------------------------|-------------|
| agg_name    | Specifies the name of the aggregate function to be commented out. | -           |
| agg_type    | Specifies the data types of the aggregation function parameters.  | -           |
| source_type | Specifies the name of the source data type of the cast.           | -           |
| target_type | Specifies the name of the target data type of the cast.           | -           |



| Parameter                                        | Description                                                                                                                                                   | Value Range                           |
|--------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
| object_name                                      | Specifies the name of an object.                                                                                                                              | -                                     |
| table_name.column_name<br>/view_name.column_name | Specifies the column whose comment is defined or modified. You can add the table name or view name as the prefix.                                             | -                                     |
| constraint_name                                  | Specifies the table constraints whose comment is defined or modified.                                                                                         | Name of an existing table constraint. |
| table_name                                       | Specifies the table name.                                                                                                                                     | Name of an existing table.            |
| function_name                                    | Specifies the function whose comment is defined or modified.                                                                                                  | Use an existing function name.        |
| argmode/argname/<br>argtype                      | Specifies the schema, name, and type of the function parameters.                                                                                              | -                                     |
| large_object_oid                                 | Specifies the OID of the large object whose comment is defined or modified.                                                                                   | -                                     |
| operator_name                                    | Specifies the name of the operator.                                                                                                                           | -                                     |
| left_type/right_type                             | Specifies the data type of the operation parameter, which can be specified by a pattern. Write NONE for the missing argument of a prefix or postfix operator. | -                                     |
| text                                             | Specifies the new comment.                                                                                                                                    | A string.                             |

## Examples

Create a sample table **CUSTOMER** and add comments to the **customer.c\_customer\_sk** column of the table.

```
CREATE TABLE IF NOT EXISTS CUSTOMER
(
 C_CUSTKEY BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY ,
 C_NAME VARCHAR(25) ,
 C_ADDRESS VARCHAR(40) ,
 C_NATIONKEY INT ,
```

```
C_PHONE CHAR(15) ,
C_ACCTBAL DECIMAL(15,2)
)
DISTRIBUTE BY HASH(C_CUSTKEY);

COMMENT ON COLUMN customer.c_customer_sk IS 'Primary key of customer demographics table.';
```

Create the **customer\_details\_view** view and add comments to the view.

```
CREATE OR REPLACE VIEW customer_details_view AS SELECT * FROM CUSTOMER;
COMMENT ON VIEW customer_details_view_v2 IS 'View of customer detail';
```

Add comments to the **customer** table.

```
COMMENT ON TABLE customer IS 'This is my table';
```

## 21.13.13 Publications and Subscriptions

### 21.13.13.1 CREATE PUBLICATION

#### Function

**CREATE PUBLICATION** adds a new publication to the current database. The publication name must be different from the name of any existing publication in the current database. A publication is essentially the replication of data changes in a set of tables achieved by logical replication.

#### Precautions

- This statement is supported by version 8.2.0.100 or later clusters.
- If neither **FOR TABLE** nor **FOR ALL TABLES** is specified, a publication starts with a set of empty tables. Tables can be added later.
- Creating a publication does not start replication. It defines only one group and filtering logic for future subscribers. To create a publication, the caller must have CREATE permission on the current database.
- To add a table to a publication, the caller must have ownership of the table. To use FOR ALL TABLES and FOR ALL TABLES IN SCHEMA clauses, the caller must have system administrator permissions.
- Do not add a table to the same publication by using FOR TABLE and FOR ALL TABLES IN SCHEMA.

#### Syntax

```
CREATE PUBLICATION name
[FOR ALL TABLES
 | FOR publication_object [, ...]]
[WITH (publication_parameter [=value] [, ...])];
```

The syntax of using **publication\_object** is as follows:

```
TABLE table_name [, ...]
| ALL TABLES IN SCHEMA schema_name [, ...]
```

## Parameter Description

**Table 21-269** CREATE PUBLICATION parameters

| Parameter                                              | Description                                                                                                                               | Value Range                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name                                                   | Specifies the name of a new publication.                                                                                                  | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                      |
| FOR ALL TABLES                                         | Marks a publication as replications of all fine-grained DR primary table changes in the database, including tables to be created.         | -                                                                                                                                                                                                                                                                                                                                                             |
| FOR TABLE                                              | Specifies the list of tables to be added to a publication. Only the fine-grained DR primary table can be a part of the publication.       | -                                                                                                                                                                                                                                                                                                                                                             |
| table_name                                             | Name of the table to be added to the publication, which can include the schema name.                                                      | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                      |
| FOR ALL TABLES IN SCHEMA                               | Marks a publication as replications of all fine-grained DR primary table changes a specified schema list, including tables to be created. | -                                                                                                                                                                                                                                                                                                                                                             |
| schema_name                                            | The name of the schema to be added to the publication.                                                                                    | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                                                                                                                                      |
| WITH<br>( publication_parameter [=value]<br>[, ... ] ) | Specifies optional parameters for a publication. Parameters in the right column are supported.                                            | publish: Specifies the DML operations that will be published to the subscriber.<br>Value range: A string. Separate the operations by commas (,). The available operations are insert, update, delete, and truncate.<br><br>By default, all actions are released. Therefore, the default value of this option is <b>insert, update, delete, and truncate</b> . |

## Examples

- Create sample table **tpcds.ship\_mode\_t1**.

```
CREATE TABLE tpcds.ship_mode_t1
(
 SM_SHIP_MODE_SK INTEGER NOT NULL,
 SM_SHIP_MODE_ID CHAR(16) NOT NULL,
 SM_TYPE CHAR(30) ,
 SM_CODE CHAR(10) ,
 SM_CARRIER CHAR(20) ,
 SM_CONTRACT CHAR(20)
) WITH (ORIENTATION = COLUMN,enable_disaster_cstore='on')
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);
```

Create sample table **tpcds.customer\_address\_p1**.

```
CREATE TABLE tpcds.customer_address_p1
(
 CA_ADDRESS_SK INTEGER NOT NULL,
 CA_ADDRESS_ID CHAR(16) NOT NULL,
 CA_STREET_NUMBER CHAR(10) ,
 CA_STREET_NAME VARCHAR(60) ,
 CA_STREET_TYPE CHAR(15) ,
 CA_SUITE_NUMBER CHAR(10) ,
 CA_CITY VARCHAR(60) ,
 CA_COUNTY VARCHAR(30) ,
 CA_STATE CHAR(2) ,
 CA_ZIP CHAR(10) ,
 CA_COUNTRY VARCHAR(20) ,
 CA_GMT_OFFSET DECIMAL(5,2) ,
 CA_LOCATION_TYPE CHAR(20)
) WITH (ORIENTATION = COLUMN,enable_disaster_cstore='on')
DISTRIBUTE BY HASH(CA_ADDRESS_SK);
```

Create sample schema **myschema1**.

```
CREATE SCHEMA myschema1;
```

Create sample schema **myschema2**.

```
CREATE SCHEMA myschema2;
```

- Create a publication for all changes of two tables and two schemas.  

```
CREATE PUBLICATION mypublication FOR TABLE users, departments, ALL TABLES IN SCHEMA myschema1, myschema2;
```
- Create a publication for all changes in all tables.  

```
CREATE PUBLICATION alltables FOR ALL TABLES;
```

## Helpful Links

[ALTER PUBLICATION](#), [DROP PUBLICATION](#)

### 21.13.13.2 ALTER PUBLICATION

#### Function

**ALTER PUBLICATION** modifies the publication attributes.

#### Precautions

- This statement is supported by version 8.2.0.100 or later clusters.
- This statement can be used by the owner of a publication and the system administrator only.

- To alter the publication owner, you must also be a direct or indirect member of the new owning role, and that role must have CREATE privilege on the database.
- In a publication with **FOR ALL TABLES**, the new publication owner must be the system administrator.
- An administrator can change the owner relationship of any publication.

## Syntax

- Add objects to a publication.  
`ALTER PUBLICATION name ADD publication_object [, ...]`
- Delete objects from a publication.  
`ALTER PUBLICATION name DROP publication_object [, ...]`
- Replace the current object with a specified object.  
`ALTER PUBLICATION name SET publication_object [, ...]`
- Set publication parameters. Retain the original values of the parameters that are not mentioned.  
`ALTER PUBLICATION name SET ( publication_parameter [= value] [, ... ] )`
- Change the publication owner.  
`ALTER PUBLICATION name OWNER TO new_owner`
- Rename the publication.  
`ALTER PUBLICATION name RENAME TO new_name`

The syntax of using **publication\_object** is as follows:

```
TABLE table_name [, ...]
| ALL TABLES IN SCHEMA schema_name [, ...]
```

## Parameter Description

**Table 21-270** ALTER PUBLICATION parameters

| Parameter                                               | Description                                                                      | Value Range                                                                                                    |
|---------------------------------------------------------|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------|
| name                                                    | Specifies the publication name to be modified.                                   | -                                                                                                              |
| table_name                                              | Specifies the table name.                                                        | Name of an existing table.                                                                                     |
| schema_name                                             | Specifies the schema name.                                                       | Name of an existing schema.                                                                                    |
| SET<br>( publication_<br>parameter [= value] [, ... ] ) | Modifies the publication parameters initially set by <b>CREATE PUBLICATION</b> . | For details about the parameters, see the <a href="#">Parameter Description</a> in <b>CREATE PUBLICATION</b> . |
| new_owner                                               | Specifies the new owner of the publication.                                      | Name of an existing user.                                                                                      |
| new_name                                                | Specifies the new name of the publication.                                       | A string compliant with the <a href="#">identifier naming rules</a> .                                          |

## Examples

- Add a table to a publication.  
`ALTER PUBLICATION mypublication ADD TABLE mydata2;`
- Deletes a schema from a publication.  
`ALTER PUBLICATION mypublication DROP ALL TABLES IN SCHEMA myschema1;`
- Reset a publication object.  
`ALTER PUBLICATION mypublication SET TABLE mydata2, ALL TABLES IN SCHEMA myschema2;`
- Change the publication owner.  
`ALTER PUBLICATION mypublication OWNER TO user1;`
- Change the publication name.  
`ALTER PUBLICATION mypublication RENAME TO mypublication1;`

## Helpful Links

[CREATE PUBLICATION](#) and [DROP PUBLICATION](#)

### 21.13.13.3 DROP PUBLICATION

#### Function

**DROP PUBLICATION** deletes a publication.

#### Precautions

- This statement is supported by version 8.2.0.100 or later clusters.
- This statement can be used by the publication owner and system administrators only.

#### Syntax

```
DROP PUBLICATION [IF EXISTS] name [, ...] [CASCADE | RESTRICT]
```

#### Parameter Description

Table 21-271 DROP PUBLICATION parameters

| Parameter          | Description                                                                                                                                          | Value Range              |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| IF EXISTS          | If the specified publication does not exist, no error is thrown. Instead, a notification is reported indicating that the publication does not exist. | -                        |
| name               | Specifies the name of the publication you want to delete.                                                                                            | An existing publication. |
| CASCADE   RESTRICT | Currently, these keywords do not work because there is no dependency on publications.                                                                | -                        |

## Examples

Delete a publication.  
`DROP PUBLICATION mypublication;`

## Helpful Links

[ALTER PUBLICATION](#), [CREATE PUBLICATION](#)

### 21.13.13.4 CREATE SUBSCRIPTION

## Function

**CREATE SUBSCRIPTION** adds a new subscription to the current database. The subscription name must be different from a name of any existing subscription in the database. A subscription represents a replication for connecting to the publication side.

## Precautions

- This statement is supported by version 8.2.0.100 or later clusters.
- A subscription can be created only by the system administrator.

## Syntax

```
CREATE SUBSCRIPTION name
 CONNECTION 'conninfo'
 PUBLICATION publication_name [, ...]
 [WITH (subscription_parameter [= value] [, ...])]
```

## Parameter Description

**Table 21-272** CREATE SUBSCRIPTION parameters

| Parameter | Description                           | Value Range                                                                          |
|-----------|---------------------------------------|--------------------------------------------------------------------------------------|
| name      | Specifies the name of a subscription. | A string, which must comply with the <a href="#">identifier naming conventions</a> . |

| Parameter | Description                                                                                                                                                                                                                 | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| conninfo  | <p>Specifies the string for connecting to the publication side.</p> <p>Example: <b>host=1.1.1.1,2.2.2.2</b><br/><b>port=10000,20000</b><br/><b>dbname=postgres</b><br/><b>user=repusr1</b><br/><b>password=password</b></p> | <ul style="list-style-type: none"><li>• <b>host</b><br/>IP address of the publication side. You can specify the IP addresses of both the primary and standby nodes of the publication side. Separate the IP addresses with commas (,).</li><li>• <b>port</b><br/>The publication port cannot be the primary port. Instead, it must be the primary port number plus 1. Otherwise, the port number conflicts with the thread pool. You can specify the ports of both the primary and standby nodes of the publication side. Separate the ports with commas (,).</li></ul> <p><b>CAUTION</b><br/>The number of hosts must be the same as that of ports.</p> <ul style="list-style-type: none"><li>• <b>dbname</b><br/>Specifies the database where a publication is deployed.</li><li>• <b>user and password</b><br/>Specifies the username and password used to connect to the publication side. The user has the system administrator permission (<b>SYSADMIN</b>) or O&amp;M administrator permission (<b>OPRADMIN</b>).</li></ul> |



| Parameter        | Description                                                                                                                                  | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| publication_name | Specifies the name of the publication you want to subscribe to on the publication side. A subscription corresponds to multiple publications. | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| WITH             | Specifies the optional parameters for a subscription. The right column lists the supported parameters.                                       | <ul style="list-style-type: none"><li>• <b>enabled</b><br/>Specifies whether a subscription should be actively replicated, or whether it should be just set but not started.<br/>The value can be <b>true</b> or <b>false</b>.<br/>The default value is <b>true</b>.</li><li>• <b>create_slot</b><br/>Specifies whether a replication slot is created on the publisher.<br/>The value can be <b>true</b> or <b>false</b>.<br/>The default value is <b>true</b>.</li><li>• <b>slot_name</b><br/>Specifies the name of the replication slot.<br/>Value range: a string<br/>Default value: the subscription name</li></ul> |

## Examples

Create a subscription to tables of the publication **mypublication** on a remote server.

```
CREATE SUBSCRIPTION mysub
 CONNECTION 'host=192.168.1.50 port=5432 user=foo dbname=foodb password=*****'
 PUBLICATION mypublication;
```

## Helpful Links

[ALTER SUBSCRIPTION, DROP SUBSCRIPTION](#)

## 21.13.13.5 ALTER SUBSCRIPTION

### Function

This syntax modifies the attributes of a subscription.

### Precautions

- This statement is supported by version 8.2.0.100 or later clusters.
- Only the owner of a subscription can execute **ALTER SUBSCRIPTION**, and the new owner must be a system administrator.

### Syntax

- Update the connection information of a subscription.  
`ALTER SUBSCRIPTION name CONNECTION 'conninfo';`
- Update the name of the publication on the publisher side.  
`ALTER SUBSCRIPTION name SET PUBLICATION publication_name [ , ... ];`
- Enable a subscription.  
`ALTER SUBSCRIPTION name ENABLE;`
- Disable a subscription.  
`ALTER SUBSCRIPTION name DISABLE;`
- Set subscription parameters.  
`ALTER SUBSCRIPTION name SET ( subscription_parameter [= value] [ , ... ] );`
- Change the subscription owner.  
`ALTER SUBSCRIPTION name OWNER TO new_owner;`
- Rename the subscription.  
`ALTER SUBSCRIPTION name RENAME TO new_name;`

### Parameter Description

Table 21-273 ALTER SUBSCRIPTION parameters

| Parameter             | Description                                                                                            | Value Range                                                                                                 |
|-----------------------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| name                  | Specifies the name of the subscription to be modified.                                                 | Name of an existing subscription.                                                                           |
| CONNECTION 'conninfo' | Alters the connection attributes initially set by <b>CREATE SUBSCRIPTION</b> .                         | For details about the parameters, see <a href="#">Parameter Description</a> in <b>CREATE SUBSCRIPTION</b> . |
| publication_name      | Specifies the name of a new publication.                                                               | A string compliant with the <a href="#">identifier naming rules</a> .                                       |
| ENABLE                | Enables a previously disabled subscription and starts logical replication at the end of a transaction. | -                                                                                                           |

| Parameter                                              | Description                                                                                | Value Range                                                                                                          |
|--------------------------------------------------------|--------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| DISABLE                                                | Disables a running subscription and stops logical replication at the end of a transaction. | -                                                                                                                    |
| SET<br>( publication_parameter [= value]<br>[, ... ] ) | Modifies the publication parameters initially set by <b>CREATE PUBLICATION</b> .           | For details about the parameters, see the <a href="#">Parameter Description</a> part in <b>CREATE SUBSCRIPTION</b> . |
| new_owner                                              | Specifies the name new subscription owner.                                                 | Name of an existing user.                                                                                            |
| new_name                                               | Specifies the new name of the subscription.                                                | A string compliant with the <a href="#">identifier naming rules</a> .                                                |

## Examples

Modify subscription connection information.

```
ALTER SUBSCRIPTION mysub CONNECTION 'host=192.168.1.51 port=5432 user=foo dbname=foodb
password=xxxx';
```

## Helpful Links

[CREATE SUBSCRIPTION](#) and [DROP SUBSCRIPTION](#)

### 21.13.13.6 DROP SUBSCRIPTION

#### Function

**DROP SUBSCRIPTION** deletes a subscription.

#### Precautions

- This statement is supported by version 8.2.0.100 or later clusters.
- A subscription can be deleted by the system administrator only.

#### Syntax

```
DROP SUBSCRIPTION [IF EXISTS] name
```

## Parameter Description

Table 21-274 DROP SUBSCRIPTION parameters

| Parameter | Description                                                                                                                                            | Value Range               |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------|
| IF EXISTS | If the specified subscription does not exist, no error is thrown. Instead, a notification is reported indicating that the subscription does not exist. | -                         |
| name      | Specifies the name of the subscription you want to delete.                                                                                             | An existing subscription. |

## Examples

```
Delete a subscription.
DROP SUBSCRIPTION mysub;
```

## Helpful Links

[ALTER SUBSCRIPTION](#), [CREATE SUBSCRIPTION](#)

## 21.13.14 Other

### 21.13.14.1 CREATE SYNONYM

#### Function

**CREATE SYNONYM** is used to create a synonym object. A synonym is an alias of a database object and is used to record the mapping between database object names. You can use synonyms to access associated database objects.

#### Precautions

- The user of a synonym should be its owner.
- If the schema name is specified, create a synonym in the specified schema. Otherwise create a synonym in the current schema.
- Database objects that can be accessed using synonyms include tables, views, functions, and stored procedures.
- To use synonyms, you must have the required permissions on associated objects.
- The following DML statements support synonyms: **SELECT**, **INSERT**, **UPDATE**, **DELETE**, **EXPLAIN**, and **CALL**.
- The **CREATE SYNONYM** statement of an associated function or stored procedure cannot be used in a stored procedure. You are advised to use synonyms existing in the **pg\_synonym** system catalog in the stored procedure.

## Syntax

```
CREATE [OR REPLACE] SYNONYM synonym_name
FOR object_name;
```

## Parameter Description

**Table 21-275** CREATE SYNONYM parameters

| Parameter   | Description                                                                                                                                                | Value Range                                                                                 |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| synonym     | Name of a synonym which is created (optionally with schema names)                                                                                          | A string that must comply with the rules in <a href="#">Identifier Naming Conventions</a> . |
| object_name | Name of an object that is associated (optionally with schema names)<br><b>NOTE</b><br><b>object_name</b> can be the name of an object that does not exist. | A string that must comply with the rules in <a href="#">Identifier Naming Conventions</a> . |

## Examples

Create schema **ot**.

```
CREATE SCHEMA ot;
```

Create table **ot.t1** and its synonym **t1**.

```
CREATE TABLE ot.t1(id int, name varchar2(10)) DISTRIBUTE BY hash(id);
CREATE OR REPLACE SYNONYM t1 FOR ot.t1;
```

Use synonym **t1**.

```
SELECT * FROM t1;
INSERT INTO t1 VALUES (1, 'ada'), (2, 'bob');
UPDATE t1 SET t1.name = 'cici' WHERE t1.id = 2;
```

Create synonym **v1** and its associated view **ot.v\_t1**.

```
CREATE SYNONYM v1 FOR ot.v_t1;
CREATE VIEW ot.v_t1 AS SELECT * FROM ot.t1;
```

Use synonym **v1**.

```
SELECT * FROM v1;
```

Create overloaded function **ot.add** and its synonym **add**.

```
CREATE OR REPLACE FUNCTION ot.add(a integer, b integer) RETURNS integer AS
$$
SELECT $1 + $2
$$
LANGUAGE sql;

CREATE OR REPLACE FUNCTION ot.add(a decimal(5,2), b decimal(5,2)) RETURNS decimal(5,2) AS
$$
SELECT $1 + $2
$$
```

```
LANGUAGE sql;
CREATE OR REPLACE SYNONYM add FOR ot.add;
```

Use synonym **add**.

```
SELECT add(1,2);
SELECT add(1,2,2.3);
```

Create stored procedure **ot.register** and its synonym **register**.

```
CREATE PROCEDURE ot.register(n_id integer, n_name varchar2(10))
SECURITY INVOKER
AS
BEGIN
 INSERT INTO ot.t1 VALUES(n_id, n_name);
END;
/

CREATE OR REPLACE SYNONYM register FOR ot.register;
```

Use synonym **register** to invoke the stored procedure.

```
CALL register(3,'mia');
```

## Helpful Links

[ALTER SYNONYM](#), [DROP SYNONYM](#)

### 21.13.14.2 ALTER SYNONYM

#### Function

**ALTER SYNONYM** is used to modify the attribute of a synonym.

#### Precautions

- Only the owner of the **SYNONYM** object can be changed.
- Only the system administrator and the owner of the **SYNONYM** object can change its owner information.
- The modifier must be a direct or indirect member of the new owner, and the new owner must have the **CREATE** permission on the schema to which the synonym belongs.

#### Syntax

```
ALTER SYNONYM synonym_name
OWNER TO new_owner;
```

#### Parameter Description

**Table 21-276** ALTER SYNONYM parameters

| Parameter    | Description                                                            | Value Range                  |
|--------------|------------------------------------------------------------------------|------------------------------|
| synonym_name | Name of the synonym to be modified, which can include the schema name. | Name of an existing synonym. |

| Parameter | Description                                   | Value Range     |
|-----------|-----------------------------------------------|-----------------|
| OWNER TO  | Clause used to modify the owner of a synonym. | -               |
| new_owner | New owner of a synonym.                       | Valid username. |

## Examples

Create synonym **t1**.

```
CREATE OR REPLACE SYNONYM t1 FOR ot.t1;
```

Create user **u1**.

```
CREATE USER u1 PASSWORD '{password}';
```

Change the owner of the synonym **t1** to **u1**.

```
ALTER SYNONYM t1 OWNER TO u1;
```

## Helpful Links

[CREATE SYNONYM](#) and [DROP SYNONYM](#)

### 21.13.14.3 DROP SYNONYM

## Function

**DROP SYNONYM** is used to delete a synonym object.

## Precautions

Only a synonym owner or a system administrator can run the **DROP SYNONYM** command.

## Syntax

```
DROP SYNONYM [IF EXISTS] synonym_name [CASCADE | RESTRICT];
```

## Parameter Description

**Table 21-277** DROP SYNONYM parameters

| Parameter    | Description                                                                           | Value Range               |
|--------------|---------------------------------------------------------------------------------------|---------------------------|
| IF EXISTS    | Sends a notice instead of reporting an error if the specified synonym does not exist. | -                         |
| synonym_name | Name of a synonym which is deleted (optionally with schema names)                     | An existing synonym name. |

| Parameter          | Description                                                                                         | Value Range                                                                                                                                                                                                                                                          |
|--------------------|-----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CASCADE   RESTRICT | Specifies how to process related data in the dependent object when a delete operation is performed. | <ul style="list-style-type: none"><li>• <b>CASCADE</b>: automatically deletes objects (such as views) that depend on the synonym to be deleted.</li><li>• <b>RESTRICT</b>: refuses to delete the synonym if any objects depend on it. This is the default.</li></ul> |

## Examples

Delete a synonym.

```
DROP SYNONYM t1;
```

## Helpful Links

[ALTER SYNONYM](#) and [CREATE SYNONYM](#)

### 21.13.14.4 ALTER LARGE OBJECT

## Function

This syntax modifies the definition of a large object. The only function of ALTER LARGE OBJECT is to assign a new owner.

## Precautions

Only the owner of a large object can run this command. System administrators have this permission by default.

## Syntax

```
ALTER LARGE OBJECT large_object_oid
OWNER TO new_owner;
```

## Parameter Description

**Table 21-278** ALTER LARGE OBJECT parameters

| Parameter        | Description                                       | Value Range                       |
|------------------|---------------------------------------------------|-----------------------------------|
| large_object_oid | OID of the large object to be modified            | Name of an existing large object. |
| OWNER TO         | Clause used to modify the owner of a large object | -                                 |



| Parameter | Description             | Value Range                       |
|-----------|-------------------------|-----------------------------------|
| new_owner | Owner of a large object | Name of an existing user or role. |

## Examples

None.

### 21.13.14.5 CREATE BARRIER

#### Function

Creates a barrier for cluster nodes. The barrier can be used for data restoration.

#### Precautions

Before creating a barrier, ensure that **gtm\_backup\_barrier** and **enable\_cbm\_tracking** are set to **on** for CNs and DN nodes in the cluster.

#### Syntax

```
CREATE BARRIER [barrier_name] ;
```

#### Parameter Description

**barrier\_name**

(Optional) Indicates the name of a barrier.

A string compliant with the [identifier naming rules](#).

## Examples

Create a barrier without specifying its name.

```
CREATE BARRIER;
```

Create a barrier named **barrier1**.

```
CREATE BARRIER 'barrier1';
```

## 21.14 DML Syntax

### 21.14.1 DML Syntax Overview

Data Manipulation Language (DML) is used to perform operations on data in database tables, such as inserting, updating, querying, or deleting data.

#### Insert Data

Inserting data refers to adding one or multiple records to a database table. For details, see [INSERT](#).

## Updating Data

Modifying data refers to modifying one or multiple records in a database table. For details, see [UPDATE](#).

## Querying Data

The database query statement **SELECT** is used to search required information in a database. For details, see [SELECT](#).

## Deleting Data

For details about how to delete data that meets specified conditions from a table, see [DELETE](#).

## Copying Data

DWS provides a statement for copying data between tables and files. For details, see [COPY](#).

## Locking a Table

DWS provides multiple lock modes to control concurrent accesses to table data. For details, see [LOCK](#).

## Run the following statement to invoke the function:

DWS provides three statements for invoking functions. These statements are the same in the syntax structure. For details, see [CALL](#).

## 21.14.2 CALL

### Function

**CALL** calls defined functions or stored procedures.

### Precautions

If the name of a user-defined function is the same as that of a system function, you need to specify a schema when invoking the user-defined function. Otherwise, the system preferentially invokes the system function.

### Syntax

```
CALL [schema.] {func_name| procedure_name} (param_expr);
```

## Parameter Description

**Table 21-279** CALL parameters

| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Value Range                                                                                                              |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| schema     | Specifies the name of the schema to which a function or stored procedure belongs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| func_name  | Specifies the name of the function or stored procedure to be called.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Use an existing function name.                                                                                           |
| param_expr | <p>Specifies a list of parameters in the function. You can use ":" or "&gt;" to separate a parameter name from its value. By doing so, parameters can be arranged in any order. If the parameter list contains only parameter values, the sequence of the parameter values must be the same as that defined in the function or stored procedure.</p> <p><b>NOTE</b></p> <p>The parameters include input parameters (whose name and type are separated by <b>IN</b>) and output parameters (whose name and type are separated by <b>OUT</b>). When you run the <b>CALL</b> statement to call a function or stored procedure, the parameter list must contain an output parameter for non-overloaded functions. You can set the output parameter to a variable or any constant. For details, see <a href="#">Examples</a>. For an overloaded package function, the parameter list can have no output parameter, but the function may not be found. If an output parameter is contained, it must be a constant.</p> | Names of existing function parameters or stored procedure parameters.                                                    |

## Examples

Create the **func\_add\_sql** function to compute the sum of two integers and return the result:

```
CREATE FUNCTION func_add_sql(num1 integer, num2 integer) RETURN integer
AS
BEGIN
RETURN num1 + num2;
END;
/
```

Transfer based on parameter values:

```
CALL func_add_sql(1, 3);
```

Transfer based on the naming flags:

```
CALL func_add_sql(num1 => 1,num2 => 3);
CALL func_add_sql(num2 := 2, num1 := 3);
```

Delete a function:

```
DROP FUNCTION func_add_sql;
```

Create a function with output parameters:

```
CREATE FUNCTION func_increment_sql(num1 IN integer, num2 IN integer, res OUT integer)
RETURN integer
AS
BEGIN
res := num1 + num2;
END;
/
```

Set output parameters to constants:

```
CALL func_increment_sql(1,2,1);
```

Set output parameters to variables:

```
DECLARE
res int;
BEGIN
func_increment_sql(1, 2, res);
dbms_output.put_line(res);
END;
/
```

Create overloaded functions:

```
create or replace procedure package_func_overload(col int, col2 out int) package
as
declare
col_type text;
begin
col := 122;
dbms_output.put_line('two out parameters ' || col2);
end;
/
create or replace procedure package_func_overload(col int, col2 out varchar)
package
as
declare
col_type text;
begin
col2 := '122';
dbms_output.put_line('two varchar parameters ' || col2);
end;
/
```

Call a function:

```
call package_func_overload(1, 'test');
call package_func_overload(1, 1);
```

Delete a function:

```
DROP FUNCTION func_increment_sql;
```

## 21.14.3 COPY

### Function

**COPY** copies data between tables and files. **COPY FROM** copies data from a file to a table. **COPY TO** copies data from a table to a file.

Use **COPY FROM** and **COPY TO** for limited local data transfers with low concurrency.

---

#### NOTICE

- Since DWS is a fully managed cloud service, direct access to backend nodes is not available, making the **COPY** syntax unusable in this environment. Instead, employ the **\COPY** command via the gsql client; note that other tools like SQL editors do not support it.
  - For better results, save your data files on OBS and use OBS foreign tables for importing into the database.
- 

### Precautions

- **COPY** applies to only tables and does not apply to views.
- To insert data to a table, you must have the permission to insert data.
- If a list of columns is specified, **COPY** will only copy the data in the specified columns to or from the file. If there are any columns in the table that are not in the column list, **COPY FROM** will insert the default values for those columns.
- If a source data file is specified, the file must be accessible from the server. If **STDIN** is specified, data is transmitted between the client and server. Separate columns by pressing **Tab**. Enter **\.** in a new line to indicate the end of input.
- If the number of columns in a row of the data file is smaller or larger than the expected number, **COPY FROM** displays an error message.
- A backslash and a period (**\.**) indicate the end of data. The end identifier is not required for reading data from a file and is required for copying data between client applications.
- In **COPY FROM**, **"\N"** indicates an empty character string, and **"\\N"** indicates the actual data **"\N"**.
- **COPY FROM** does not support pre-processing of data during data import, for example, expression calculation and default value filling. If you need to perform pre-processing during data import, you need to import the data to a temporary table, and then run SQL statements to insert data to the table using expression or function operations. However, this method may cause I/O expansion, deteriorating data import performance.
- Transactions will be rolled back when data format errors occur during **COPY FROM** execution. In this case, error information is insufficient so you cannot easily locate the incorrect data from a large amount of raw data.
- You can use **COPY TO** to export data in TEXT or CSV format to OBS, but cannot use **COPY FROM** to import data from OBS. For CSV files that are

exported to OBS and contain utf8 BOM, you are advised to use OBS to import them. Otherwise, the BOM field may fail to be identified.

## Syntax

- Copy the data from a file to a table:

```
COPY table_name [(column_name [, ...])]
 FROM { 'filename' | STDIN }
 [[USING] DELIMITERS 'delimiters']
 [WITHOUT ESCAPING]
 [LOG ERRORS]
 [LOG ERRORS data]
 [REJECT LIMIT 'limit']
 [[WITH] (option [, ...])]
 | copy_option
 | FIXED FORMATTER ({ column_name(offset, length) } [, ...]) [(option [, ...]) | copy_option
 [...]]];
```

### NOTE

In the SQL syntax, **FIXED, FORMATTER ( { column\_name( offset, length ) } [, ...] )**, and **[ ( option [, ...] ) | copy\_option [ ...] ]** can be in any sequence.

- Copy the data from a table to a file:

```
COPY table_name [(column_name [, ...])]
 TO { 'filename' | STDOUT }
 [[USING] DELIMITERS 'delimiters']
 [WITHOUT ESCAPING]
 [[WITH] (option [, ...])]
 | copy_option
 | FIXED FORMATTER ({ column_name(offset, length) } [, ...]) [(option [, ...]) | copy_option
 [...]]];
```

```
COPY query
 TO { 'filename' | STDOUT }
 [WITHOUT ESCAPING]
 [[WITH] (option [, ...])]
 | copy_option
 | FIXED FORMATTER ({ column_name(offset, length) } [, ...]) [(option [, ...]) | copy_option
 [...]]];
```

### NOTE

- The syntax constraints of **COPY TO** are as follows:  
(**query**) is incompatible with **[USING] DELIMITER**. If the data of **COPY TO** comes from a query result, **COPY TO** cannot specify **[USING] DELIMITERS**.
- Use spaces to separate **copy\_option** following **FIXED FORMATTER**.
- **copy\_option** is the native parameter, while **option** is the parameter imported by a compatible foreign table.
- In the SQL syntax, **FIXED, FORMATTER ( { column\_name( offset, length ) } [, ...] )**, and **[ ( option [, ...] ) | copy\_option [ ...] ]** can be in any sequence.

## Parameter Description

**Table 21-280** COPY parameters

| Parameter   | Description                                                                                                                                                                                                                                                                                                                 | Value Range                                                                                                                                                                                                                                                                                                          |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| query       | Copies the query result.                                                                                                                                                                                                                                                                                                    | A <b>SELECT</b> or <b>VALUES</b> command in parentheses.                                                                                                                                                                                                                                                             |
| table_name  | Specifies the name (optionally schema-qualified) of an existing table.                                                                                                                                                                                                                                                      | An existing table name.                                                                                                                                                                                                                                                                                              |
| column_name | Specifies the list of optional fields to be copied.                                                                                                                                                                                                                                                                         | If the column list is not specified, all columns in the table will be copied.                                                                                                                                                                                                                                        |
| STDIN       | Indicates that the input comes from the client application.                                                                                                                                                                                                                                                                 | -                                                                                                                                                                                                                                                                                                                    |
| STDOUT      | Specifies that output goes to the client application.                                                                                                                                                                                                                                                                       | -                                                                                                                                                                                                                                                                                                                    |
| FIXED       | Fixes column length. <ul style="list-style-type: none"><li>• When the column length is fixed, <b>DELIMITER</b>, <b>NULL</b>, and <b>CSV</b> cannot be specified.</li><li>• When <b>FIXED</b> is specified, <b>BINARY</b>, <b>CSV</b>, and <b>TEXT</b> cannot be specified by <b>option</b> or <b>copy_option</b>.</li></ul> | The definition of fixed length: <ul style="list-style-type: none"><li>• The column length of each record is the same.</li><li>• Spaces are added to short columns. Digit type columns must be left-aligned, and character columns must be right-aligned.</li><li>• No delimiters are used between columns.</li></ul> |

| Parameter                            | Description                                                                                                  | Value Range                                                                                                                                                                               |
|--------------------------------------|--------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [USING]<br>DELIMITER<br>'delimiters' | The string that separates columns within each row (line) of the file, and it cannot be larger than 10 bytes. | The delimiter cannot include any of the following characters:<br>\\abcdefghijklmnopqrstuvwxyz0123456789<br>The default value is a tab character in text format and a comma in CSV format. |
| WITHOUT<br>ESCAPING                  | Specifies that the backslash (\) and its following characters are not escaped in text format.                | Only the text format is supported.                                                                                                                                                        |



| Parameter  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range                                                 |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| LOG ERRORS | <p>If this parameter is specified, the error tolerance mechanism for data type errors in the <b>COPY FROM</b> statement is enabled. Row errors are recorded in the <b>public.pgxc_copy_error_log</b> table in the database for future reference.</p> <p><b>NOTICE</b></p> <p>The restrictions of this error tolerance parameter are as follows:</p> <ul style="list-style-type: none"><li>• This error tolerance mechanism captures only the data type errors (DATA_EXCEPTION) that occur during data parsing of <b>COPY FROM</b> on a CN. Other errors, such as network errors between CNs and DN or expression conversion errors on DNs, are not captured.</li><li>• Before enabling error tolerance for <b>COPY FROM</b> for the first time in a database, check whether the <b>public.pgxc_copy_error_log</b> table exists. If it does not, call the <code>copy_error_log_create()</code> function to create it. If it does, copy its data elsewhere and call the <code>copy_error_log_create()</code> function to create the table. For details about columns in the <b>public.pgxc_copy_error_log</b> table, see <a href="#">Table 21-60</a>.</li><li>• While a <b>COPY FROM</b> statement with specified <b>LOG ERRORS</b> is being executed, if <b>public.pgxc_copy_error_log</b> does not exist or does not have the table definitions compliant with the predefined in <code>copy_error_log_create()</code>, an error will be reported. Ensure that the error table is created using the <code>copy_error_log_create()</code> function. Otherwise, <b>COPY FROM</b> statements with error tolerance may fail to be run.</li><li>• If existing error tolerance parameters (for example, <b>IGNORE_EXTRA_DATA</b>) of the <b>COPY</b> statement are enabled, the error of the corresponding type will be processed as specified by the parameters and no error will be reported. Therefore, the error table does not contain such error data.</li><li>• The coverage scope of this error tolerance mechanism is the same as that of a GDS foreign table. You are advised to filter query results based on table names or the timestamp of marking the start of <b>COPY FROM</b> statement execution. For how to process error data, see the section about handling error tables.</li></ul> | A value set while data is imported using <b>COPY FROM</b> . |

| Parameter                       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Value Range                                                                                                                                                                                                                                                                                                                         |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOG ERRORS DATA                 | The differences between <b>LOG ERRORS DATA</b> and <b>LOG ERRORS</b> are as follows: <ul style="list-style-type: none"><li>• <b>LOG ERRORS DATA</b> records error row data (<b>rawrecord</b>).</li><li>• When this parameter is used, if the error content is too complex, the error may fail to be written into the fault tolerance table. As a result, the task fails.</li></ul>                                                                                                                                                                                                           | -                                                                                                                                                                                                                                                                                                                                   |
| REJECT LIMIT 'limit'            | Used with the <b>LOG ERROR</b> parameter to set the upper limit of the tolerated errors in the <b>COPY FROM</b> statement. If the number of errors exceeds the limit, later errors will be reported based on the original mechanism.<br><b>NOTE</b><br>Different from the GDS error tolerance mechanism, in the error tolerance mechanism described in the description of <b>LOG ERRORS</b> , the count of <b>REJECT LIMIT</b> is calculated based on the number of data parsing errors on the CN where the <b>COPY FROM</b> statement is run, not based on the number of errors on each DN. | A positive integer (1 to INTMAX) or <b>unlimited</b><br>Default value: <ul style="list-style-type: none"><li>• If <b>LOG ERRORS</b> is not specified, an error is reported.</li><li>• If <b>LOG ERRORS</b> is specified, the default value is <b>0</b>.</li></ul>                                                                   |
| FORMATTER                       | Defines the position of each column in the data file in the <b>column(offset,length)</b> format when the fixed length mode is used (that is, the <b>FIXED</b> parameter is specified).                                                                                                                                                                                                                                                                                                                                                                                                       | <ul style="list-style-type: none"><li>• The value of <b>offset</b> must be larger than 0. The unit is byte.</li><li>• The value of <b>length</b> must be larger than 0. The unit is byte.</li><li>• The total length of all columns must be less than 1 GB.</li><li>• Replace columns that are not in the file with NULL.</li></ul> |
| OPTION { option_name 'value ' } | Specifies all types of parameters of a compatible foreign table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | For details, see <a href="#">Table 21-281</a> .                                                                                                                                                                                                                                                                                     |

| Parameter                                   | Description                                               | Value Range                                     |
|---------------------------------------------|-----------------------------------------------------------|-------------------------------------------------|
| COPY_OPTION<br>{ option_name<br>' value ' } | Specifies all types of native parameters of <b>COPY</b> . | For details, see <a href="#">Table 21-282</a> . |

**Table 21-281** OPTION parameters

| Parameter | Description                                                                                                                                                                    | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FORMAT    | Specifies the format of the source data file in the foreign table.                                                                                                             | <p>CSV, text, fixed, and binary files are supported.</p> <ul style="list-style-type: none"><li>• The CSV file can process newline characters efficiently, but cannot process certain special characters well.</li><li>• The TEXT file can process special characters efficiently, but cannot process newline character well.</li><li>• The FIXED file can process newline characters in data columns efficiently, but cannot process special characters well.</li><li>• All data in the BINARY file is stored/read as binary format rather than as text. It is faster than the text and CSV formats, but a binary-format file is less portable.</li></ul> <p>Default value: <b>TEXT</b></p> |
| OIDS      | <p>Copies the OID for each row.</p> <p><b>NOTE</b></p> <p>An error is raised if OIDs are specified for a table that does not have OIDs, or in the case of copying a query.</p> | <ul style="list-style-type: none"><li>• true/on</li><li>• false/off</li></ul> <p>Default value: <b>false</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Value Range                                                                                                                                                                                                                                                                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DELIMITER | <p>Specifies the character that separates columns within each row (line) of the file.</p> <ul style="list-style-type: none"><li>• A delimiter cannot be \r or \n.</li><li>• A delimiter cannot be the same as null. The delimiter for CSV cannot be same as quote.</li><li>• The delimiter for the TEXT format data cannot contain lowercase letters, digits, or dot (.).</li><li>• The data length of a single row should be less than 1 GB. If the delimiters are too long and there are too many rows, the length of valid data will be affected.</li><li>• You are advised to use multi-characters and invisible characters for delimiters. For example, you can use multi-characters (such as \$^&amp;) and invisible characters (such as 0x07, 0x08, and 0x1b).</li><li>• For a multi-character delimiter, do not use the same characters, for example, ---.</li></ul> <p><b>NOTE</b></p> | <p>The value of <b>delimiter</b> can be a multi-character delimiter whose length is less than or equal to 10 bytes.</p> <p>Default value:</p> <ul style="list-style-type: none"><li>• A tab character in TEXT format</li><li>• A comma (,) in CSV format</li><li>• No delimiter in FIXED format</li></ul>                                                             |
| NULL      | <p>Specifies how to represent a null value.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <ul style="list-style-type: none"><li>• The null value cannot be \r or \n. The maximum length is 100 characters.</li><li>• The null value cannot be the same as the delimiter or quote parameter.</li></ul> <p>Default value:</p> <ul style="list-style-type: none"><li>• An empty string without quotation marks in CSV format</li><li>• \N in TEXT format</li></ul> |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Value Range                                                                                                                                                                                                                                                                                                                                            |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| HEADER    | <p>Specifies whether a file contains a header with the names of each column in the file. header is available only for CSV and FIXED files.</p> <ul style="list-style-type: none"><li>When data is imported, if <b>header</b> is <b>on</b>, the first row of the data file will be identified as title row and ignored. If header is <b>off</b>, the first row is identified as data.</li><li>When data is exported, if <b>header</b> is <b>on</b>, <b>fileheader</b> must be specified. If header is <b>off</b>, the exported file does not include a title row.</li></ul> | <ul style="list-style-type: none"><li>true/on</li><li>false/off</li></ul> <p>Default value: <b>false</b></p>                                                                                                                                                                                                                                           |
| QUOTE     | <p>Specifies the quote character used when a data value is referenced in a CSV file.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | <p>Default value: double quotation mark (").</p> <ul style="list-style-type: none"><li>The quote parameter cannot be the same as the delimiter or null parameter.</li><li>The <b>quote</b> parameter must be a single one-byte character.</li><li>Invisible characters are recommended as <b>quote</b> values, such as 0x07, 0x08, and 0x1b.</li></ul> |
| ESCAPE    | <p>Specifies the escape character in CSV format. The escape character can only be a single-byte character.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <p>Single-byte character.</p> <p>Default value: double quotation mark (")</p>                                                                                                                                                                                                                                                                          |

| Parameter                                      | Description                                                                                                                                | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EOL<br>'newline_character'                     | Specifies the newline character style of the imported or exported data file.                                                               | <p>Multi-character newline characters are supported, but the newline character cannot exceed 10 bytes.</p> <p>Common newline characters include <code>\r</code> (0x0D), <code>\n</code> (0x0A), and <code>\r\n</code> (0x0D0A). Special newline characters include <code>\$</code> and <code>#</code>.</p> <ul style="list-style-type: none"><li>• The <b>EOL</b> parameter supports only the TEXT format for data import and export and does not support the CSV or FIXED format for data import. For forward compatibility, the <b>EOL</b> parameter can be set to <b>0x0D</b> or <b>0x0D0A</b> for data export in the CSV and FIXED formats.</li><li>• The value of the <b>EOL</b> parameter cannot be the same as that of <b>DELIMITER</b> or <b>NULL</b>.</li><li>• The <b>EOL</b> parameter value cannot contain lowercase letters, digits, or dot (.).</li></ul> |
| FORCE_QUOTE<br>{ ( column_name [, ...] )   * } | Forces quotation marks to be used for all non-null values in each specified column in CSV COPY TO mode. <b>NULL</b> values are not quoted. | Existing fields.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| FORCE_NOT_NULL<br>( column_name [, ...] )      | Assigns a value to a specified column in <b>CSV COPY FROM</b> mode.                                                                        | Existing fields.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

| Parameter         | Description                                                                                                                                                                                         | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ENCODING          | Specifies the encoding format of a data file.                                                                                                                                                       | <p>The default value is the current database encoding format.</p> <p>Common encoding formats include UTF-8, GBK, and GB18030. GB18030 has two versions: GB18030 and GB18030_2022. GB18030_2022 is the latest national standard in China prepared to support Chinese characters.</p>                                                                                                                                                                                                                                                                                                                                                                            |
| IGNORE_EXTRA_DATA | When the number of data source files exceeds the number of foreign table columns, whether ignoring excessive columns at the end of the row. This parameter is available only during data importing. | <ul style="list-style-type: none"><li>When this parameter is <b>true</b> or <b>on</b> and the number of data source files exceeds the number of foreign table columns, excessive columns will be ignored.</li><li>If the parameter is set to <b>false</b> or <b>off</b>, and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:<br/>extra data after last expected column</li></ul> <p>Default value: <b>false</b></p> <p><b>NOTICE</b></p> <p>If the newline character at the end of the row is lost, setting the parameter to <b>true</b> will ignore data in the next row.</p> |

| Parameter                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Value Range                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| COMPATIBLE_ILLEGAL_CHARS | <p>Specifies the error tolerance parameter for invalid characters. This parameter is valid only for <b>COPY FROM</b> import.</p> <p>This parameter does not support the binary format. The "cannot specify bulkload compatibility options in BINARY mode" error will be reported.</p> <p>The rule of error tolerance when you import invalid characters is as follows:</p> <ul style="list-style-type: none"><li>• \0 is converted to a space.</li><li>• Other invalid characters are converted to question marks.</li><li>• Setting <b>compatible_illegal_chars</b> to <b>true/on</b> enables toleration of invalid characters. If <b>NULL</b>, <b>DELIMITER</b>, <b>QUOTE</b>, and <b>ESCAPE</b> are set to spaces or question marks, errors like "illegal chars conversion may confuse COPY escape 0x20" will be displayed to prompt the user to modify parameters that may cause confusion, preventing importing errors.</li></ul> | <ul style="list-style-type: none"><li>• When the parameter is <b>true</b> or <b>on</b>, invalid characters are tolerated and imported to the database after conversion.</li><li>• If the parameter is <b>false</b> or <b>off</b>, and an error occurs when there are invalid characters, the import will be interrupted.</li></ul> <p>Default value: <b>false</b> or <b>off</b></p> |
| FILL_MISSING_FIELDS      | <p>Specifies whether to generate an error message when the last column in a row in the source file is lost during data loading.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | <ul style="list-style-type: none"><li>• <b>true/on</b>: The value of the last field is set to <b>NULL</b> and no error message is reported.</li><li>• <b>false/off</b>: The following error message is reported:<br/>missing data for column "tt"</li></ul> <p>Default value: <b>false</b></p>                                                                                      |



| Parameter                   | Description                                                                                                                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DATE_FORMAT<br>AT           | Imports data of the <b>DATE</b> type. It is valid only for <b>COPY FROM</b> import.<br><br>This parameter does not support the binary format. The "cannot specify bulkload compatibility options in BINARY mode" error will be reported.                                          | The value must be in a valid DATE format. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .<br><br><b>NOTICE</b><br>In databases configured for Oracle compatibility (with <b>DBCOMPATIBILITY</b> set to <b>ORA</b> during table creation), the DATE type functions as <b>TIMESTAMP</b> . For more information about the syntax compatibility differences, see "Syntax Compatibility Differences Among Oracle, Teradata, and MySQL". For details, see <b>timestamp_format</b> below. |
| TIME_FORMAT<br>AT           | Imports data of the <b>TIME</b> type. This parameter is valid only for <b>COPY FROM</b> import.<br><br>The <b>BINARY</b> format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur.          | Valid <b>TIME</b> formats. Time zones are not supported. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                                                                                                                                                                                                                                                                                                                                                                           |
| TIMESTAMP_FORMAT            | Imports data of the <b>TIMESTAMP</b> type. This parameter is valid only for <b>COPY FROM</b> import.<br><br>The <b>BINARY</b> format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur.     | Valid <b>TIMESTAMP</b> formats. Time zones are not supported. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                                                                                                                                                                                                                                                                                                                                                                      |
| SMALLDATE_TIME_FORMAT<br>AT | Imports data of the <b>SMALLDATETIME</b> type. This parameter is valid only for <b>COPY FROM</b> import.<br><br>The <b>BINARY</b> format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. | Valid <b>SMALLDATETIME</b> formats. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                                                                                                                                                                                                                                                                                                                                                                                                |

| Parameter       | Description                                                                                                                                                                                                                                                                                                                                                                                  | Value Range                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SERVER          | Specifies the OBS server, which takes effect only when <b>COPY TO</b> is used. <b>filename</b> is a path on OBS, indicating that data is exported to OBS.                                                                                                                                                                                                                                    | Name of the created OBS server.                                                                                                                                                                                                                                                                                                                                                                                 |
| BOM             | Specifies whether to add the UTF-8 BOM field to the exported CSV file. This parameter is valid only when <b>COPY TO</b> is executed and the <b>SERVER</b> parameter is set to a valid value. The exported file must be encoded in UTF-8 format.                                                                                                                                              | <ul style="list-style-type: none"><li>• true/on</li><li>• false/off</li></ul> Default value: <b>false</b>                                                                                                                                                                                                                                                                                                       |
| MAXROW          | Specifies the maximum number of lines in an exported file. A new file is generated if the number of lines exceeds the maximum. This parameter is valid only when <b>COPY TO</b> is executed and the <b>SERVER</b> parameter is set to a valid value. When <b>HEADER</b> is set to <b>true</b> , <b>MAXROW</b> must be greater than 1. It must be specified together with <b>FILEPREFIX</b> . | 1-2147483647                                                                                                                                                                                                                                                                                                                                                                                                    |
| FILEPREFIX      | Specifies the prefix of an export file name. This parameter is valid only when <b>COPY TO</b> is executed and the <b>SERVER</b> parameter is set to a valid value. It must be specified together with <b>MAXROW</b> .                                                                                                                                                                        | A valid string that does not start or end with a slash (/).                                                                                                                                                                                                                                                                                                                                                     |
| PRESERVE_BLANKS | Specifies whether to retain the blank characters (including spaces, \t, \v, and \f) at the end of each column during fixed-length import. This parameter is supported only by clusters of 8.2.0.100 or later.                                                                                                                                                                                | <ul style="list-style-type: none"><li>• <b>true/on</b>: indicates that the trailing \t is retained and the column data of the BYTEAOID, CHAROID, NAMEOID, TEXTOID, BPCHAROID, VARCHAROID, NVARCHAR2OID and CSTRINGOID types is not pruned.</li><li>• <b>false/off</b>: prunes trailing white space characters (including spaces, \t, \v, and \f) of all data types.</li></ul> Default value: <b>false/off</b> . |

**Table 21-282** COPY\_OPTION parameters

| Parameter           | Description                                                                                                                                                                                                                                                               | Value Range                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OIDS                | Copies the OID for each row.<br>An error is raised if OIDs are specified for a table that does not have OIDs, or in the case of copying a query.                                                                                                                          | -                                                                                                                                                                                                                                                                                                                                                                                                                        |
| NULL<br>null_string | Specifies the string that represents a null value.<br><b>NOTICE</b><br>When using <b>COPY FROM</b> , any data item that matches this string will be stored as a <b>NULL</b> value, so you should make sure that you use the same string as you used with <b>COPY TO</b> . | <ul style="list-style-type: none"><li>• The null value cannot be \r or \n. The maximum length is 100 characters.</li><li>• The null value cannot be the same as the delimiter or quote parameter.</li></ul> Default value: <ul style="list-style-type: none"><li>• \N in TEXT format</li><li>• An empty string without quotation marks in CSV format</li></ul>                                                           |
| HEADER              | Specifies whether a file contains a header with the names of each column in the file. header is available only for CSV and FIXED files.                                                                                                                                   | <ul style="list-style-type: none"><li>• When data is imported, if <b>header</b> is <b>on</b>, the first row of the data file will be identified as title row and ignored. If header is <b>off</b>, the first row is identified as data.</li><li>• When data is exported, if header is <b>on</b>, <b>fileheader</b> must be specified. If header is <b>off</b>, the exported file does not include a title row.</li></ul> |

| Parameter                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | Value Range      |
|--------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| FILEHEADER                                 | <p>Specifies a file that defines the content in the header for exported data. The file contains data description of each column.</p> <p><b>NOTICE</b></p> <ul style="list-style-type: none"><li>• This parameter is available only when <b>header</b> is <b>on</b> or <b>true</b>.</li><li>• <b>fileheader</b> specifies an absolute path.</li><li>• The file can contain only one row of header information, and ends with a linefeed. Excess rows will be discarded. (Header information cannot contain linefeeds.)</li><li>• The length of the file including the linefeed cannot exceed 1 MB.</li></ul>                                                                                                                                                   | -                |
| FREEZE                                     | <p>Sets the <b>COPY</b> loaded data row as <b>frozen</b>, like these data have executed <b>VACUUM FREEZE</b>.</p> <p>This is a performance option of initial data loading. The data will be frozen only when the following three requirements are met:</p> <ul style="list-style-type: none"><li>• The table being loaded has been created or truncated in the current subtransaction before copying.</li><li>• There are no cursors open in the current transaction.</li><li>• There are no original snapshots in the current transaction.</li></ul> <p><b>NOTICE</b></p> <p>When <b>COPY</b> is completed, all other sessions will immediately recognize the data. This violates the general principle of MVCC visibility and leads to potential risks.</p> | -                |
| FORCE NOT NULL<br>column_name [, ...]      | Assigns a value to a specified column in <b>CSV COPY FROM</b> mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Existing fields. |
| FORCE QUOTE<br>{ column_name [, ...]   * } | Forces quotation marks to be used for all non-null values in each specified column in <b>CSV COPY TO</b> mode. NULL values are not enclosed in quotation marks.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Existing fields. |

| Parameter                      | Description                                                                                                                                                                                                                                                                                                                                               | Value Range                                                                                                                     |
|--------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| BINARY                         | The binary format option causes all data to be stored/read as binary format rather than as text. In binary mode, you cannot declare <b>DELIMITER</b> , <b>NULL</b> , or <b>CSV</b> . After the binary type is specified, you cannot use <b>option</b> or <b>copy_option</b> to specify the CSV, fixed, and text types.                                    | -                                                                                                                               |
| CSV                            | Enables the CSV mode. After the CSV type is specified, you cannot use <b>option</b> or <b>copy_option</b> to specify the binary, fixed, or text type.                                                                                                                                                                                                     | -                                                                                                                               |
| QUOTE [AS] 'quote_character'   | Specifies the quote character for a CSV file. <ul style="list-style-type: none"><li>• The quote parameter cannot be the same as the delimiter or null parameter.</li><li>• The <b>quote</b> parameter must be a single one-byte character.</li><li>• Invisible characters are recommended as <b>quote</b> values, such as 0x07, 0x08, and 0x1b.</li></ul> | Default value: double quotation mark (")                                                                                        |
| ESCAPE [AS] 'escape_character' | Specifies an escape character for a CSV file. The value must be a single-byte character.                                                                                                                                                                                                                                                                  | The default value is a double quotation mark ("). If it is the same as the value of <b>quote</b> , it will be replaced with \0. |

| Parameter                   | Description                                                                                                                                                                                                                                                                                                                                                                                                               | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EOL<br>'newline_character'  | Specifies the newline character style of the imported or exported data file.                                                                                                                                                                                                                                                                                                                                              | <p>Multi-character newline characters are supported, but the newline character cannot exceed 10 bytes.</p> <p>Common newline characters include <code>\r</code> (0x0D), <code>\n</code> (0x0A), and <code>\r\n</code> (0x0D0A). Special newline characters include <code>\$</code> and <code>#</code>.</p> <ul style="list-style-type: none"><li>• The <b>EOL</b> parameter supports only the TEXT format for data import and export. For forward compatibility, the <b>EOL</b> parameter can be set to <b>0x0D</b> or <b>0x0D0A</b> for data export in the CSV and FIXED formats.</li><li>• The value of the <b>EOL</b> parameter cannot be the same as that of <b>DELIMITER</b> or <b>NULL</b>.</li><li>• The <b>EOL</b> parameter value cannot contain lowercase letters, digits, or dot (.).</li></ul> |
| ENCODING<br>'encoding_name' | Specifies the name of a file encoding format.                                                                                                                                                                                                                                                                                                                                                                             | Valid encoding formats.<br>Default value: current encoding format.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| IGNORE_EXTRA_DATA           | <p>When the number of data source files exceeds the number of foreign table columns, excess columns at the end of the row are ignored. This parameter is available only during data importing.</p> <p>If you do not use this parameter, and the number of data source files exceeds the number of foreign table columns, the following error information will be displayed:<br/>extra data after last expected column</p> | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

| Parameter                | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Value Range                                                                                                                    |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| COMPATIBLE_ILLEGAL_CHARS | <p>Specifies error tolerance for invalid characters during importing. Invalid characters are converted before importing. No error is reported and the import is not interrupted. The binary format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. It is valid only for <b>COPY FROM</b> import.</p> <p>If this parameter is not used, an error is reported when an invalid character is encountered during the import, and the import is interrupted.</p> <p>The rule of error tolerance when you import invalid characters is as follows:</p> <ul style="list-style-type: none"><li>• \0 is converted to a space.</li><li>• Other invalid characters are converted to question marks.</li><li>• Setting <b>compatible_illegal_chars</b> to <b>true/on</b> enables toleration of invalid characters. If <b>NULL</b>, <b>DELIMITER</b>, <b>QUOTE</b>, and <b>ESCAPE</b> are set to spaces or question marks, errors like "illegal chars conversion may confuse COPY escape 0x20" will be displayed to prompt the user to modify parameters that may cause confusion, preventing importing errors.</li></ul> | -                                                                                                                              |
| FILL_MISSING_FIELDS      | <p>Specifies whether to generate an error message when the last column in a row in the source file is lost during data loading.</p> <p><b>NOTICE</b><br/>Do not specify this option. If this option is enabled, the parser will ignore the data problem during CN data parsing (that is, the fault tolerance range of the COPY error table) and report an error again on the DN. As a result, the COPY error table (with <b>LOG ERRORS REJECT LIMIT</b> enabled) cannot capture such data exceptions.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | <ul style="list-style-type: none"><li>• true/on</li><li>• false/off</li></ul> <p>Default value: <b>false</b> or <b>off</b></p> |

| Parameter                                                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Value Range                                                                                                                                       |
|-----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| DATE_FORM<br>AT<br>'date_format_string'                   | <p>Imports data of the <b>DATE</b> type. The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur. This parameter is valid only for <b>COPY FROM</b> import.</p> <p><b>NOTE</b><br/>In databases configured for Oracle compatibility (with <b>DBCOMPATIBILITY</b> set to <b>ORA</b> during table creation), the DATE type functions as <b>TIMESTAMP</b>. For more information about the syntax compatibility differences, see "Syntax Compatibility Differences Among Oracle, Teradata, and MySQL". For details, see <b>timestamp_format</b> below.</p> | Valid DATE formats. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                                           |
| TIME_FORM<br>AT<br>'time_format_string'                   | <p>Imports data of the <b>TIME</b> type. This parameter is valid only for <b>COPY FROM</b> import.</p> <p>The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur.</p>                                                                                                                                                                                                                                                                                                                                                                                 | Valid TIME formats. Time zones are not supported. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .             |
| TIMESTAMP_FORMAT<br>'timestamp_format_string'             | <p>Imports data of the <b>TIMESTAMP</b> type. This parameter is valid only for <b>COPY FROM</b> import.</p> <p>The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur.</p>                                                                                                                                                                                                                                                                                                                                                                            | Valid <b>TIMESTAMP</b> formats. Time zones are not supported. For details, see <a href="#">Date and Time Processing Functions and Operators</a> . |
| SMALLDATETIME_FORM<br>AT<br>'smalldatetime_format_string' | <p>Imports data of the <b>SMALLDATETIME</b> type. This parameter is valid only for <b>COPY FROM</b> import.</p> <p>The BINARY format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur.</p>                                                                                                                                                                                                                                                                                                                                                                        | Valid <b>SMALLDATETIME</b> formats. For details, see <a href="#">Date and Time Processing Functions and Operators</a> .                           |



| Parameter       | Description                                                                                                                                                                                                      | Value Range                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PRESERVE_BLANKS | Specifies whether to retain the blank characters (including spaces, \t, \v, and \f) at the end of each column during fixed-length import. This parameter is supported by clusters of version 8.2.0.100 or later. | <ul style="list-style-type: none"><li>• <b>true/on</b>: indicates that the trailing \t is retained and the column data of the BYTEAOID, CHAROID, NAMEOID, TEXTOID, BPCHAROID, VARCHAROID, NVARCHAR2OID and CSTRINGOID types is not pruned.</li><li>• <b>false/off</b>: prunes trailing white space characters (including spaces, \t, \v, and \f) of all data types.</li></ul> Default value: <b>false/off</b> . |

The following special backslash sequences are recognized by **COPY FROM**:

**Table 21-283** Special backslash sequences

| Backslash Sequence | Description                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------|
| \b                 | Backslash (ASCII 8)                                                                                                |
| \f                 | Form feed (ASCII 12)                                                                                               |
| \n                 | Newline character (ASCII 10)                                                                                       |
| \r                 | Carriage return character (ASCII 13)                                                                               |
| \t                 | Tab (ASCII 9)                                                                                                      |
| \v                 | Vertical tab (ASCII 11)                                                                                            |
| \digits            | Backslash followed by one to three octal digits specifies the ASCII value is the character with that numeric code. |
| \xdigits           | Backslash followed by an x and one or two hex digits specifies the character with that numeric code.               |

### Example (supported only by the gsql client)

Copy data from the **tpcds.ship\_mode** file to the **/home/omm/ds\_ship\_mode.dat** file:

```
\COPY tpcds.ship_mode TO '/home/omm/ds_ship_mode.dat';
```

Copy the query result from the **tpcds.ship\_mode** file to the **/home/omm/ds\_ship\_mode.dat** file.

```
\COPY (SELECT * FROM tpcds.ship_mode WHERE a <> null) TO '/home/omm/ds_ship_mode.dat';
```

Write **tpcds.ship\_mode** as output to **stdout**:

```
\COPY tpcds.ship_mode TO stdout;
```

Create the **tpcds.ship\_mode\_t1** table:

```
CREATE TABLE tpcds.ship_mode_t1
(
 SM_SHIP_MODE_SK INTEGER NOT NULL,
 SM_SHIP_MODE_ID CHAR(16) NOT NULL,
 SM_TYPE CHAR(30) ,
 SM_CODE CHAR(10) ,
 SM_CARRIER CHAR(20) ,
 SM_CONTRACT CHAR(20)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE)
DISTRIBUTE BY HASH(SM_SHIP_MODE_SK);
```

Copy data from **stdin** to the **tpcds.ship\_mode\_t1** table:

```
\COPY tpcds.ship_mode_t1 FROM stdin;
```

Copy data from the **/home/omm/ds\_ship\_mode.dat** file to the **tpcds.ship\_mode\_t1** table:

```
\COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat';
```

Copy data from the **/home/omm/ds\_ship\_mode.dat** file to the **tpcds.ship\_mode\_t1** table, with the import format set to TEXT (**format 'text'**), the delimiter set to **\t** (delimiter **E'\t'**), excessive columns ignored (**ignore\_extra\_data 'true'**), and characters not escaped (**noescaping 'true'**).

```
\COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' WITH(format 'text', delimiter E'\t',
ignore_extra_data 'true', noescaping 'true');
```

Copy data from the **/home/omm/ds\_ship\_mode.dat** file to the **tpcds.ship\_mode\_t1** table, with the import format set to FIXED, fixed-length format specified (**FORMATTER(SM\_SHIP\_MODE\_SK(0, 2), SM\_SHIP\_MODE\_ID(2,16), SM\_TYPE(18,30), SM\_CODE(50,10), SM\_CARRIER(61,20), SM\_CONTRACT(82,20))**), excessive columns ignored (**ignore\_extra\_data**), and headers included (**header**).

```
\COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' FIXED
FORMATTER(SM_SHIP_MODE_SK(0, 2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10),
SM_CARRIER(61,20), SM_CONTRACT(82,20)) header ignore_extra_data;
```

Copy data from the **/home/omm/ds\_ship\_mode.dat** file to the **tpcds.ship\_mode\_t1** table, with the import format set to FIXED, fixed-length format specified (**FORMATTER(SM\_SHIP\_MODE\_SK(0, 2), SM\_SHIP\_MODE\_ID(2,16), SM\_TYPE(18,30), SM\_CODE(50,10), SM\_CARRIER(61,20), SM\_CONTRACT(82,20))**), excessive columns ignored (**ignore\_extra\_data**), headers included (**header**), and trailing **\t** retained.

```
\COPY tpcds.ship_mode_t1 FROM '/home/omm/ds_ship_mode.dat' (FORMAT 'fixed', FORMATTER
```

```
(SM_SHIP_MODE_SK(0,2), SM_SHIP_MODE_ID(2,16), SM_TYPE(18,30), SM_CODE(50,10),
SM_CARRIER(61,20), SM_CONTRACT(82,20)), PRESERVE_BLANKS>true', HEADER 'true',
IGNORE_EXTRA_DATA 'true');
```

Export **tpcds.ship\_mode\_t1** as a text file **ds\_ship\_mode.dat** in the OBS directory /**bucket/path/**. You need to specify the server options that contain OBS access information:

```
COPY tpcds.ship_mode_t1 TO '/bucket/path/ds_ship_mode.dat' WITH (format 'text', encoding 'utf8', server
'obs_server');
```

Export **tpcds.ship\_mode\_t1** as a CSV file in the OBS directory /**bucket/path/**. You need to specify the server options that contain OBS access information. The file contains the title line and BOM header. A single file can contain a maximum of 1000 lines. If the number of lines exceeds 1000, a new file is generated. The user-defined file name prefix is **justprefix**:

```
COPY (select * from tpcds.ship_mode_t1 where SM_SHIP_MODE_SK=1060) TO '/bucket/path/' WITH (format
'csv', header 'on', encoding 'utf8', server 'obs_server', bom 'on', maxrow '1000', fileprefix 'justprefix');
```

Delete the **tpcds.ship\_mode\_t1** table:

```
DROP TABLE tpcds.ship_mode_t1;
```

## 21.14.4 DELETE

### Function

Deletes rows that satisfy the **WHERE** clause from the specified table. If the **WHERE** clause does not exist, all rows in the table will be deleted. The result is a valid, but an empty table.

### Precautions

- You must have the **DELETE** permission on the table to delete from it, as well as the **SELECT** permission for any table in the **USING** clause or whose values are read in the **condition**.
- For replication tables, **DELETE** can be performed only in the following two scenarios:
  - Scenarios with primary key constraints.
  - Scenario where the execution plan can be pushed down.
- For column-store tables, the **RETURNING** clause is currently not supported.

---

#### WARNING

- Avoid using **UPDATE** or **DELETE** to modify or delete a large volume of data and use **TRUNCATE PARTITION** or **DROP PARTITION** instead.
  - For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Data Warehouse Service (DWS) Developer Guide*.
- 

### Syntax

```
[WITH [RECURSIVE] with_query [...]]
DELETE [/*+ plan_hint */] FROM [ONLY] table_name [*] [[AS] alias]
```

```
[PARTITION (partition_name) | PARTITION FOR (partition_key_value [, ...])]
[USING using_list]
[WHERE condition | WHERE CURRENT OF cursor_name]
[RETURNING { * | { output_expr [[AS] output_name] } [, ...] }];
```

## Parameter Description

Table 21-284 DELETE parameters

| Parameter                                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range             |
|------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| WITH<br>[ RECURSIVE ]<br>with_query<br>[, ...] | <p>The <b>WITH</b> clause allows you to specify one or more subqueries that can be referenced by name in the primary query, equal to temporary table.</p> <p>If <b>RECURSIVE</b> is specified, it allows a <b>SELECT</b> subquery to reference itself by name.</p> <p>The format of <b>with_query</b> is as follows:<br/>with_query_name [ ( column_name [, ...] ) ] AS<br/>( {select   values   insert   update   delete} )</p> <p>-- <b>with_query_name</b> specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of</p> <p><b>column_name</b> specifies the column name displayed in the subquery result set.</p> <p>Each subquery can be a <b>SELECT</b>, <b>VALUES</b>, <b>INSERT</b>, <b>UPDATE</b> or <b>DELETE</b> statement.</p> | -                       |
| plan_hint<br>clause                            | Following the keyword in the <b>/*+ */</b> format, hints are used to optimize the plan generated by a specified statement block. For details, see "DWS Performance Tuning" > "SQL Tuning" > "Hint-based Tuning" in <i>Data Warehouse Service (DWS) Developer Guide</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | -                       |
| ONLY                                           | <p>If <b>ONLY</b> is specified, only the specified table is deleted. If <b>ONLY</b> is not specified, the specified table and all its inherited tables are deleted.</p> <p>This parameter is reserved only for compatibility with PostgreSQL. DWS does not support inherited tables.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | -                       |
| table_name                                     | Specifies the name (optionally schema-qualified) of a target table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | An existing table name. |

| Parameter                    | Description                                                                                                                                                                                      | Value Range                                                                                                              |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| alias                        | Specifies the alias for the target table.                                                                                                                                                        | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| partition_name               | Specifies the name of a partition. Only clusters of version 8.2.1 or later support this option.                                                                                                  | An existing partition name.                                                                                              |
| partition_key_value          | Specifies the key value of a partition.<br>The value specified by <b>PARTITION FOR ( partition_key_value [, ...] )</b> can uniquely identify a partition.                                        | Value range of the partition key for the partition to be renamed.                                                        |
| using_list                   | Specifies the <b>USING</b> clause.                                                                                                                                                               | -                                                                                                                        |
| condition                    | Specifies an expression that returns a value of type boolean. Only rows for which this expression returns <b>true</b> will be deleted.                                                           | -                                                                                                                        |
| WHERE CURRENT OF cursor_name | Not supported currently. Only syntax interface is provided.                                                                                                                                      | -                                                                                                                        |
| output_expr                  | Specifies an expression to be computed and returned by the <b>DELETE</b> command after each row is deleted. The expression can use any column names of the table. Write * to return all columns. | -                                                                                                                        |
| output_name                  | Specifies a name to use for a returned column.                                                                                                                                                   | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |

## Examples

Create the **tpcds.customer\_address\_bak** table:

```
CREATE TABLE tpcds.customer_address_bak AS TABLE tpcds.customer_address;
```

Delete employees whose **ca\_address\_sk** is less than **14888** in the **tpcds.customer\_address\_bak** table:

```
DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk < 14888;
```

Delete the employees whose **ca\_address\_sk** is **14891**, **14893**, and **14895** from **tpcds.customer\_address\_bak**:

```
DELETE FROM tpcds.customer_address_bak WHERE ca_address_sk in (14891,14893,14895);
```

Delete all data in the **tpcds.customer\_address\_bak** table:

```
DELETE FROM tpcds.customer_address_bak;
```

Use a subquery (to delete the row-store table **tpcds.warehouse\_t30**) to obtain a temporary table **temp\_t**, and then query all data in the temporary table **temp\_t**:

```
WITH temp_t AS (DELETE FROM tpcds.warehouse_t30 RETURNING *) SELECT * FROM temp_t ORDER BY 1;
```

Delete partition **p1** from the partitioned table **test\_range\_row**:

```
CREATE TABLE test_range_row(a int, d int)
DISTRIBUTE BY hash(a) PARTITION BY RANGE(d)
(
 PARTITION p1 values LESS THAN (60),
 PARTITION p2 values LESS THAN (75),
 PARTITION p3 values LESS THAN (90),
 PARTITION p4 VALUES LESS THAN (maxvalue)
);
INSERT OVERWRITE INTO test_range_row PARTITION(p1) VALUES(55,51);
INSERT OVERWRITE INTO test_range_row PARTITION(p3) VALUES(85,80);

DELETE FROM test_range_row PARTITION(p1);
```

## 21.14.5 EXPLAIN

### Function

**EXPLAIN** outputs the execution plan of SQL statements. You can analyze the execution plan to locate slow SQL statements and optimize SQL statements.

The execution plan shows how the tables referenced by the SQL statement will be scanned, for example, by plain sequential scan or index scan. If multiple tables are referenced, the execution plan also shows what join algorithms will be used to bring together the required rows from each input table. The most critical part of the display is the estimated statement execution cost, which is the planner's guess at how long it will take to run the statement.

The **ANALYZE** option (**EXPLAIN ANALYZE**) causes the statement to be actually executed, not only planned. The actual execution time of each step is displayed. This helps you verify the accuracy of the execution plan and evaluate the actual performance.

### Syntax

- Display the execution plan of an SQL statement, which supports multiple options and has no requirements for the order of options.

```
EXPLAIN [(option [, ...])] statement;
```

The syntax of the **option** clause is as follows:

```
ANALYZE [boolean] |
ANALYSE [boolean] |
VERBOSE [boolean] |
COSTS [boolean] |
CPU [boolean] |
DETAIL [boolean] |
NODES [boolean] |
NUM_NODES [boolean] |
```

```
BUFFERS [boolean] |
TIMING [boolean] |
PLAN [boolean] |
FORMAT { TEXT | XML | JSON | YAML } |
BLOCKNAME [boolean] |
OUTLINE [boolean] |
WARMUP |
WARMUP HOT
```

- Display the execution plan of an SQL statement, where options are in order.  
`EXPLAIN { [ { ANALYZE | ANALYSE } ] [ VERBOSE ] | PERFORMANCE } statement;`
- Display information required for reproducing the execution plan of an SQL statement. The information is usually used for fault locating. The **STATS** option must be used independently.  
`EXPLAIN ( STATS [ boolean ] ) statement;`
- Display the detailed time consumption of the DDL statement execution. This syntax is supported only by clusters of version 9.1.0 or later.  
`EXPLAIN PERFORMANCE statement;`  
DDL statement such as CREATE, CREATE INDEX, DROP, VACUUM FULL, ANALYZE, and COPY are supported.
- Perform pre-query and cache the pre-queried data to the local disk for higher query speed. This syntax is supported only by clusters of 9.1.0.200 and later versions.  
`EXPLAIN WARMUP statement;`  
`EXPLAIN WARMUP HOT statement;`

## Parameter Description

Table 21-285 EXPLAIN parameters

| Parameter                                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Value Range                                                                                                                                                                       |
|--------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| statement                                  | Specifies the SQL statement to be analyzed.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | -                                                                                                                                                                                 |
| ANALYZE<br>boolean  <br>ANALYSE<br>boolean | <p>The <b>ANALYZE</b> option causes the statement to be executed, not only planned. Then actual runtime statistics are added to the display, including the total elapsed time expended within each plan node (in milliseconds) and the total number of rows it actually returned.</p> <p>To use <b>EXPLAIN ANALYZE</b> on an <b>INSERT</b>, <b>UPDATE</b>, <b>DELETE</b>, <b>CREATE TABLE AS</b>, or <b>EXECUTE</b> statement without letting the command affect your data, you can place it in a transaction and roll back it after the analysis is complete.</p> <pre>START TRANSACTION;<br/>EXPLAIN ANALYZE ...;<br/>ROLLBACK;</pre> | <ul style="list-style-type: none"><li>• <b>TRUE</b>: displays the actual run times and other statistics.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b> |

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                | Value Range                                                                                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VERBOSE<br>boolean   | Displays additional information about the query plan.<br>Additional information includes the output column list for each node in the plan tree, schema-qualify table and function names, always label variables in expressions with their range table alias, and always print the name of each trigger for which statistics are displayed. | <ul style="list-style-type: none"><li>• <b>TRUE</b>: The information is displayed.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b>            |
| COSTS boolean        | Displays the estimated total cost of each plan node, as well as the estimated number of rows and the estimated width of each row.                                                                                                                                                                                                          | <ul style="list-style-type: none"><li>• <b>TRUE</b>: The information is displayed.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b>            |
| CPU boolean          | Displays CPU usage.                                                                                                                                                                                                                                                                                                                        | <ul style="list-style-type: none"><li>• <b>TRUE</b>: The CPU usage is displayed.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b>              |
| DETAIL<br>boolean    | Displays DN information.<br><b>NOTE</b><br>For clusters of 8.2.1 or later, when <b>Detail</b> is enabled in <b>EXPLAIN</b> , the time required for comparing skew values is displayed in the execution plan.                                                                                                                               | <ul style="list-style-type: none"><li>• <b>TRUE</b>: prints DN information.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b>                   |
| NODES<br>boolean     | Prints information about the nodes executed by query.                                                                                                                                                                                                                                                                                      | <ul style="list-style-type: none"><li>• <b>TRUE</b>: prints information about executed nodes.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b> |
| NUM_NODES<br>boolean | Prints the quantity of executing nodes.                                                                                                                                                                                                                                                                                                    | <ul style="list-style-type: none"><li>• <b>TRUE</b>: prints the number of DNs.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b>                |



| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range                                                                                                                                                                                                                                                                                                                                      |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BUFFERS<br>boolean   | Displays information about the buffer usage.<br><br>The buffer information includes the number of shared blocks hit, read, and updated, the number of local blocks hit, read, and updated, the number of temp blocks hit, read, and updated. Shared blocks contain data from regular tables and indexes, local blocks contain data from temporary tables and indexes, and temporary blocks contain short-term working data used in sorts, hashes. | <ul style="list-style-type: none"><li>• <b>TRUE</b>: displays information on buffer usage.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>FALSE</b>                                                                                                                                                                             |
| TIMING<br>boolean    | Displays the actual startup time and time spent in each node in the output.                                                                                                                                                                                                                                                                                                                                                                       | <ul style="list-style-type: none"><li>• <b>TRUE</b>: displays the information.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b>                                                                                                                                                                                          |
| PLAN boolean         | Specifies whether to store the execution plan in the <b>plan_table</b> table. When this option is enabled, the execution plan is stored in <b>PLAN_TABLE</b> and is not displayed on the current screen. Therefore, this option cannot be used together with other options when it is enabled.                                                                                                                                                    | <ul style="list-style-type: none"><li>• <b>ON</b>: The execution plan is stored in <b>plan_table</b> and is not displayed on the current screen. If the plan is stored successfully, <b>EXPLAIN SUCCESS</b> is returned.</li><li>• <b>OFF</b>: The execution plan is not stored but is printed on the screen.</li></ul> Default value: <b>ON</b> |
| FORMAT               | Specifies the output format.<br><br>The output content of each format is the same. XML, JSON, and YAML are easier for programs to parse SQL statements.                                                                                                                                                                                                                                                                                           | Text, XML, JSON, and YAML formats are supported.<br><br>Default value: <b>TEXT</b>                                                                                                                                                                                                                                                               |
| BLOCKNAME<br>boolean | Displays the blockname information of an operator.<br><br>The blockname information is printed only when <b>explain_perf_mode</b> is set to <b>pretty</b> .                                                                                                                                                                                                                                                                                       | <ul style="list-style-type: none"><li>• <b>TRUE</b>: displays blockname information.</li><li>• <b>FALSE</b>: displays no blockname information.</li></ul> Default value: <b>TRUE</b>                                                                                                                                                             |

| Parameter          | Description                                                                                                                                                                                                                              | Value Range                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OUTLINE<br>boolean | Displays outline information extracted from the plan.<br>The outline is printed only when <b>explain_perf_mode</b> is set to <b>pretty</b> .                                                                                             | <ul style="list-style-type: none"><li>• <b>TRUE</b>: displays outline information.</li><li>• <b>FALSE</b>: displays no outline information.</li></ul> Default value: <b>TRUE</b>                                   |
| PERFORMANCE        | This option prints all relevant information in execution.                                                                                                                                                                                | -                                                                                                                                                                                                                  |
| STATS boolean      | Specifies whether to display information required for reproducing the execution plan of an SQL statement, including the object definition, statistics, and configuration parameters. The information is usually used for fault locating. | <ul style="list-style-type: none"><li>• <b>TRUE</b>: display information required for reproducing the execution plan of an SQL statement.</li><li>• <b>FALSE</b>: No display.</li></ul> Default value: <b>TRUE</b> |
| WARMUP             | Processes the queried data in the sequence of A1in > A1out > Am. This is supported only by clusters of version 9.1.0.200 or later.                                                                                                       | -                                                                                                                                                                                                                  |
| WARMUP HOT         | Directly adds the queried data to the Am column. This is supported only by clusters of version 9.1.0.200 or later.                                                                                                                       | -                                                                                                                                                                                                                  |

## Examples

Create the **tpcds.customer\_address\_p1** table.

```
CREATE TABLE tpcds.customer_address_p1 AS TABLE tpcds.customer_address;
```

Change the value of **explain\_perf\_mode** to **normal**.

```
SET explain_perf_mode=normal;
```

Display an execution plan for simple queries in the table.

```
EXPLAIN SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN

Data Node Scan on "__REMOTE_FQS_QUERY__" (cost=0.00..0.00 rows=0 width=0)
Node/s: All datanodes
(2 rows)
```

Generate an execution plan in JSON format (assume **explain\_perf\_mode** is set to **normal**).

```
EXPLAIN(FORMAT JSON) SELECT * FROM tpcds.customer_address_p1;
QUERY PLAN

[
 {
 "Plan": {
 +
 +
 +
```

```

 "Node Type": "Data Node Scan", +
 "RemoteQuery name": "__REMOTE_FQS_QUERY__",+
 "Alias": "__REMOTE_FQS_QUERY__", +
 "Startup Cost": 0.00, +
 "Total Cost": 0.00, +
 "Plan Rows": 0, +
 "Plan Width": 0, +
 "Nodes": "All datanodes" +
 } +
 } +
]
(1 row)

```

If there is an index and we use a query with an indexable **WHERE** condition, **EXPLAIN** might show a different plan.

```

EXPLAIN SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN

Data Node Scan on "__REMOTE_LIGHT_QUERY__" (cost=0.00..0.00 rows=0 width=0)
Node/s: datanode2
(2 rows)

```

Generate an execution plan in YAML format (assume **explain\_perf\_mode** is set to **normal**).

```

EXPLAIN(FORMAT YAML) SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN

- Plan:
 Node Type: "Data Node Scan"
 RemoteQuery name: "__REMOTE_LIGHT_QUERY__"
 Alias: "__REMOTE_LIGHT_QUERY__"
 Startup Cost: 0.00
 Total Cost: 0.00
 Plan Rows: 0
 Plan Width: 0
 Nodes: "datanode2"
(1 row)

```

Here is an example of an execution plan with cost estimates suppressed.

```

EXPLAIN(COSTS FALSE)SELECT * FROM tpcds.customer_address_p1 WHERE ca_address_sk=10000;
QUERY PLAN

Data Node Scan on "__REMOTE_LIGHT_QUERY__"
Node/s: datanode2
(2 rows)

```

Here is an example of an execution plan for a query that uses an aggregate function.

```

EXPLAIN SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE ca_address_sk<10000;
QUERY PLAN

Aggregate (cost=18.19..14.32 rows=1 width=4)
-> Streaming (type: GATHER) (cost=18.19..14.32 rows=3 width=4)
 Node/s: All datanodes
 -> Aggregate (cost=14.19..14.20 rows=3 width=4)
 -> Seq Scan on customer_address_p1 (cost=0.00..14.18 rows=10 width=4)
 Filter: (ca_address_sk < 10000)
(6 rows)

```

-- Delete the **tpcds.customer\_address\_p1** table.

```
DROP TABLE tpcds.customer_address_p1;
```

Run the **EXPLAIN PERFORMANCE** command for the **ANALYZE** statement.

```
EXPLAIN PERFORMANCE ANALYZE t2_dist_row;
QUERY EXEC INFO

lock FirstCN:
 coordinator1: actual time=0.240 loops=1
estimate rows: actual time=[datanode3 0.000, datanode1 0.001]
 coordinator1: actual time=0.000 loops=1
 datanode1: actual time=0.001 loops=1
 datanode2: actual time=0.000 loops=1
 datanode3: actual time=0.000 loops=1
sample rows: actual time=[datanode1 5.109, coordinator1 119.838]
 coordinator1: actual time=119.838 loops=1
 datanode1: actual time=5.109 loops=1
 datanode2: actual time=5.621 loops=1
 datanode3: actual time=5.342 loops=1
fetch global stats:
 coordinator1: actual time=8.501 loops=1
calc stats: actual time=[datanode3 80.794, datanode2 109.155]
 coordinator1: actual time=97.452 loops=1
 datanode1: actual time=94.375 loops=1
 datanode2: actual time=109.155 loops=1
 datanode3: actual time=80.794 loops=1
calc column stats: actual time=[datanode2 0.938, datanode3 9.811]
 coordinator1: actual time=5.162 loops=2
 datanode1: actual time=1.453 loops=2
 datanode2: actual time=0.938 loops=2
 datanode3: actual time=9.811 loops=2
calc index stats: actual time=[datanode3 12.392, coordinator1 36.113]
 coordinator1: actual time=36.113 loops=1
 datanode1: actual time=15.933 loops=1
 datanode2: actual time=13.419 loops=1
 datanode3: actual time=12.392 loops=1
calc expr stats: actual time=[datanode3 41.665, datanode2 78.442]
 coordinator1: actual time=55.608 loops=1
 datanode1: actual time=63.179 loops=1
 datanode2: actual time=78.442 loops=1
 datanode3: actual time=41.665 loops=1
sync stats:
 coordinator1: actual time=7.906 loops=1

General Tracks
CN build CN connection:
 coordinator1: actual time=0.002 loops=1
CN build DN connection:
 coordinator1: actual time=0.070 loops=1
-> execute ddl on other CN:
 coordinator1: actual time=0.001 loops=1
-> execute ddl on other DN:
 coordinator1: actual time=0.000 loops=1
Query Id: 72902018968225366
Total runtime: 242.211 ms
(48 rows)
```

Show the blockname of a plan.

```
EXPLAIN (BLOCKNAME ON) SELECT SUM(ca_address_sk) FROM tpcds.customer_address_p1 WHERE
ca_address_sk<10000;
```

```
QUERY PLAN

id | operation | E-rows | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----
1 | -> Aggregate | 1 | | 12 | 16.14
2 | -> Streaming (type: GATHER) | 2 | | 12 | 16.14
3 | -> Aggregate | 2 | 1MB | 12 | 10.14
4 | -> Seq Scan on customer_address_p1 | 7 | 1MB | 4 | 10.12

Predicate Information (identified by plan id)

4 --Seq Scan on customer_address_p1
Filter: (ca_address_sk < 10000)
```

Query Block Name / Object Alias (identified by plan id)

```
1 - sel$1
4 - sel$1 / customer_address_p1@"sel$1"
```

===== Query Summary =====

System available mem: 4710400KB  
Query Max mem: 4710400KB  
Query estimated mem: 2048KB  
(22 rows)

Show the outline of a plan.

```
EXPLAIN (OUTLINE ON) SELECT SUM(ca_address_sk) FROM tpods.customer_address_p1 WHERE
ca_address_sk<10000;
```

#### QUERY PLAN

| id | operation                          | E-rows | E-memory | E-width | E-costs |
|----|------------------------------------|--------|----------|---------|---------|
| 1  | -> Aggregate                       | 1      |          | 12      | 16.14   |
| 2  | -> Streaming (type: GATHER)        | 2      |          | 12      | 16.14   |
| 3  | -> Aggregate                       | 2      | 1MB      | 12      | 10.14   |
| 4  | -> Seq Scan on customer_address_p1 | 7      | 1MB      | 4       | 10.12   |

Predicate Information (identified by plan id)

```
4 --Seq Scan on customer_address_p1
 Filter: (ca_address_sk < 10000)
```

#### Outline Data

```
/*+
 begin_outline_data
 TableScan(@"sel$1" tpods.customer_address_p1@"sel$1")
 end_outline_data
*/
```

===== Query Summary =====

System available mem: 4710400KB  
Query Max mem: 4710400KB  
Query estimated mem: 2048KB  
(25 rows)

## Helpful Links

[ANALYZE](#) | [ANALYZE](#)

## 21.14.6 EXPLAIN PLAN

### Function

You can run the **EXPLAIN PLAN** statement to save the information about an execution plan to the **PLAN\_TABLE** table. Different from the **EXPLAIN** statement, **EXPLAIN PLAN** only stores plan information and does not print it on the screen.

### Syntax

```
EXPLAIN PLAN
[SET STATEMENT_ID = string]
FOR statement ;
```

## Parameter Description

**Table 21-286** EXPLAIN PLAN parameters

| Parameter    | Description                                                                                                                                                                                                                                                                                                                                 |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PLAN         | Stores plan information in <b>PLAN_TABLE</b> . If the storing is successful, <b>EXPLAIN SUCCESS</b> is returned.                                                                                                                                                                                                                            |
| STATEMENT_ID | Tags a query. The tag information will be stored in <b>PLAN_TABLE</b> .<br><b>NOTE</b><br>If the <b>EXPLAIN PLAN</b> statement does not contain <b>SET STATEMENT_ID</b> , the value of <b>STATEMENT_ID</b> is empty by default. In addition, the value of <b>STATEMENT_ID</b> cannot exceed 30 bytes. Otherwise, an error will be reported. |

## Precautions

- **EXPLAIN PLAN** cannot be executed on DNs.
- Plan information cannot be collected for SQL statements that failed to be executed.
- Data in **PLAN\_TABLE** is in a session-level life cycle. Sessions are isolated from users and thereby users can view data of only the current session and current user.
- **PLAN\_TABLE** cannot be joined with GDS foreign tables.
- For a query that cannot be pushed down, object information cannot be collected and only such information as **REMOTE\_QUERY** and **CTE** can be collected. For details, see [Example 2](#).

## Example 1

You can perform the following steps to collect execution plans of SQL statements by running **EXPLAIN PLAN**:

**Step 1** Import TPC-H sample data.

**Step 2** Run the **EXPLAIN PLAN** statement.

### NOTE

After the **EXPLAIN PLAN** statement is executed, plan information is automatically stored in **PLAN\_TABLE**. **INSERT**, **UPDATE**, and **ANALYZE** cannot be performed on **PLAN\_TABLE**.

For details about **PLAN\_TABLE**, see the **PLAN\_TABLE** system view.

```
explain plan set statement_id='TPCH-Q4' for
select
o_orderpriority,
count(*) as order_count
from
orders
where
o_orderdate >= '1993-07-01'::date
and o_orderdate < '1993-07-01'::date + interval '3 month'
and exists (
select
```

```

*
from
lineitem
where
l_orderkey = o_orderkey
and l_commitdate < l_receiptdate
)
group by
o_orderpriority
order by
o_orderpriority;

```

**Step 3 Query PLAN\_TABLE.**

```
SELECT * FROM PLAN_TABLE;
```

| statement_id | plan_id  | id | operation           | options     | object_name | object_type | object_owner | projection                                    |
|--------------|----------|----|---------------------|-------------|-------------|-------------|--------------|-----------------------------------------------|
| TPCH-Q4      | 16781167 | 1  | ROW ADAPTER         |             |             |             |              | ORDERS.O_ORDERPRIORITY, (PG_CATALOG.COUNT(*)) |
| TPCH-Q4      | 16781167 | 2  | VECTOR SORT         |             |             |             |              | ORDERS.O_ORDERPRIORITY, (PG_CATALOG.COUNT(*)) |
| TPCH-Q4      | 16781167 | 3  | VECTOR AGGREGATE    | HASHED      |             |             |              | ORDERS.O_ORDERPRIORITY, PG_CATALOG.COUNT(*)   |
| TPCH-Q4      | 16781167 | 4  | VECTOR STREAMING    | GATHER      |             |             |              | ORDERS.O_ORDERPRIORITY, (COUNT(*))            |
| TPCH-Q4      | 16781167 | 5  | VECTOR AGGREGATE    | HASHED      |             |             |              | ORDERS.O_ORDERPRIORITY, COUNT(*)              |
| TPCH-Q4      | 16781167 | 6  | VECTOR NESTED LOOPS | SEMI        |             |             |              | ORDERS.O_ORDERPRIORITY                        |
| TPCH-Q4      | 16781167 | 7  | TABLE ACCESS        | CSTORE SCAN | ORDERS      | TABLE       | TPCH         | ORDERS.O_ORDERPRIORITY, ORDERS.O_ORDERKEY     |
| TPCH-Q4      | 16781167 | 8  | VECTOR MATERIALIZE  |             |             |             |              | LINEITEM.L_ORDERKEY                           |
| TPCH-Q4      | 16781167 | 9  | TABLE ACCESS        | CSTORE SCAN | LINEITEM    | TABLE       | TPCH         | LINEITEM.L_ORDERKEY                           |

**Step 4 Delete data from PLAN\_TABLE.**

```
DELETE FROM PLAN_TABLE WHERE xxx;
```

----End

**Example 2**

For a query that cannot be pushed down, only such information as **REMOTE\_QUERY** and **CTE** can be collected from **PLAN\_TABLE** after **EXPLAIN PLAN** is executed.

The optimizer generates a plan for pushing down statements. In this case, only **REMOTE\_QUERY** can be collected.

```

explain plan set statement_id = 'test remote query' for
select
current_user
from
customer;

```

**Query PLAN\_TABLE.**

```
SELECT * FROM PLAN_TABLE;
```

| statement_id      | plan_id  | id | operation      | options   | object_name      | object_type  | object_owner | projection    |
|-------------------|----------|----|----------------|-----------|------------------|--------------|--------------|---------------|
| test remote query | 29360133 | 1  | NESTED LOOPS   | CARTESIAN |                  |              |              | 'apple'::name |
| test remote query | 29360133 | 2  | DATA NODE SCAN |           | customer         | REMOTE_QUERY |              |               |
| test remote query | 29360133 | 3  | DATA NODE SCAN |           | customer_address | REMOTE_QUERY |              |               |

(3 rows)

**21.14.7 LOCK****Function**

**LOCK TABLE** obtains a table-level lock.

When the lock for commands referencing a table is automatically acquired, DWS always uses the lock mode with minimum constraints. Use **LOCK** if users need a more strict lock mode. For example, suppose an application runs a transaction at the **Read Committed** isolation level and needs to ensure that data in a table remains stable in the duration of the transaction. To achieve this, you could obtain **SHARE** lock mode over the table before the query. This will prevent concurrent

data changes and ensure subsequent reads of the table see a stable view of committed data. It is because the **SHARE** lock mode conflicts with the **ROW EXCLUSIVE** lock acquired by writers, and your **LOCK TABLE name IN SHARE MODE** statement will wait until any concurrent holders of **ROW EXCLUSIVE** mode locks commit or roll back. Therefore, once you obtain the lock, there are no uncommitted writes outstanding; furthermore none can begin until you release the lock.

## Precautions

- **LOCK TABLE** is useless outside a transaction block: the lock would remain held only to the completion of the statement. If **LOCK TABLE** is out of any transaction block, an error is reported.
- If no lock mode is specified, then **ACCESS EXCLUSIVE**, the most restrictive mode, is used.
- **LOCK TABLE ... IN ACCESS SHARE MODE** requires the **SELECT** permission on the target table. All other forms of **LOCK** require table-level **UPDATE** and/or the **DELETE** permission.
- There is no **UNLOCK TABLE** command. Locks are always released at transaction end.
- **LOCK TABLE** only deals with table-level locks, and so the mode names involving **ROW** are all misnomers. These mode names should generally be read as indicating the intention of the user to acquire row-level locks within the locked table. Also, **ROW EXCLUSIVE** mode is a shareable table lock. Keep in mind that all the lock modes have identical semantics so far as **LOCK TABLE** is concerned, differing only in the rules about which modes conflict with which. For details about the rules, see [Lock Levels and Conflicts](#).

## Syntax

```
LOCK [TABLE] {[ONLY] name [, ...] {name [*]} [, ...]}
 [IN {ACCESS SHARE | ROW SHARE | ROW EXCLUSIVE | SHARE UPDATE EXCLUSIVE | SHARE | SHARE
ROW EXCLUSIVE | EXCLUSIVE | ACCESS EXCLUSIVE | UPDATE EXCLUSIVE} MODE]
 [NOWAIT] [LOCAL COORDINATOR ONLY];
```

## Lock Levels and Conflicts

DWS uses lock modes to control how concurrent transactions access resources, ensuring data consistency and preventing loss.

[Table 21-287](#) shows the eight main lock modes and their potential conflicts.

**Table 21-287** Conflicts between lock modes

| Lock Mode       | Lock Level | Lock Usage                                                  | Conflict |
|-----------------|------------|-------------------------------------------------------------|----------|
| AccessShareLock | 1          | SELECT statement, allowing other transactions to read data. | 8        |



| Lock Mode                | Lock Level | Lock Usage                                                                                                                                                                                           | Conflict        |
|--------------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| RowShareLock             | 2          | SELECT FOR UPDATE or FOR SHARE, allowing other transactions to read but preventing writes.                                                                                                           | 7 8             |
| RowExclusiveLock         | 3          | INSERT, UPDATE, and DELETE                                                                                                                                                                           | 5 6 7 8         |
| ShareUpdateExclusiveLock | 4          | VACUUM (non-FULL), ANALYZE, CREATE INDEX CONCURRENTLY, and COMMENT ON statements allows other transactions to read data but blocks writes.                                                           | 4 5 6 7 8       |
| ShareLock                | 5          | CREATE INDEX (non-CONCURRENTLY) allows other transactions to read data, but they cannot write data.                                                                                                  | 3 4 6 7 8       |
| ShareRowExclusiveLock    | 6          | Similar to RowExclusiveLock, this lock lets you read data safely by using <b>ROW SELECT...FOR UPDATE</b> . But it permits RowShareLock, allowing others to read the row but not update or delete it. | 3 4 5 6 7 8     |
| ExclusiveLock            | 7          | This lock prevents <b>RowShareLock</b> or <b>SELECT... FOR UPDATE</b> operations.                                                                                                                    | 2 3 4 5 6 7 8   |
| AccessExclusive-Lock     | 8          | The lock stops other transactions from reading and writing data during <b>ALTER TABLE</b> , <b>DROP TABLE</b> , or <b>VACUUM FULL</b> operations.                                                    | 1 2 3 4 5 6 7 8 |

## Parameter Description

Table 21-288 LOCK parameters

| Parameter | Description                                                                                                                                                           | Value Range             |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|
| name      | The name (optionally schema-qualified) of an existing table to lock.<br><br>The tables are locked one-by-one in the order specified in the <b>LOCK TABLE</b> command. | An existing table name. |

| Parameter              | Description                                                                                                                                                                                                                                                                                                                                                                                   | Value Range |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| ONLY                   | If <b>ONLY</b> is specified, only the specified table is deleted. If <b>ONLY</b> is not specified, the specified table and all its inherited tables are deleted.<br><br>This parameter is reserved only for compatibility with PostgreSQL. DWS does not support inherited tables.                                                                                                             | -           |
| ACCESS SHARE           | <b>ACCESS SHARE</b> allows only read operations on a table. In general, any SQL statements that only read a table and do not modify it will acquire this lock mode. The <b>SELECT</b> command acquires a lock of this mode on referenced tables.                                                                                                                                              | -           |
| ROW SHARE              | <b>ROW SHARE</b> allows concurrent read of a table but does not allow any other operations on the table.<br><b>SELECT FOR UPDATE</b> and <b>SELECT FOR SHARE</b> automatically acquire the <b>ROW SHARE</b> lock on the target table and add the <b>ACCESS SHARE</b> lock to other referenced tables except <b>FOR SHARE</b> and <b>FOR UPDATE</b> .                                          | -           |
| ROW EXCLUSIVE          | Like <b>ROW SHARE</b> , <b>ROW EXCLUSIVE</b> allows concurrent read of a table and modification of data in the table. <b>UPDATE</b> , <b>DELETE</b> , and <b>INSERT</b> automatically acquire this lock on the target table and add the <b>ACCESS SHARE</b> lock to other referenced tables. Generally, all commands that modify table data acquire the <b>ROW EXCLUSIVE</b> lock for tables. | -           |
| SHARE UPDATE EXCLUSIVE | This mode protects a table against concurrent schema changes and <b>VACUUM</b> runs.<br><br>Acquired by <b>VACUUM</b> (without <b>FULL</b> ), <b>ANALYZE</b> and <b>CREATE INDEX CONCURRENTLY</b> statements.                                                                                                                                                                                 | -           |

| Parameter           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                   | Value Range                                                                                                |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| SHARE               | <b>SHARE</b> allows concurrent queries of a table but does not allow modification of the table.<br><br>Acquired by CREATE INDEX (without CONCURRENTLY).                                                                                                                                                                                                                                                                                       | -                                                                                                          |
| SHARE ROW EXCLUSIVE | <b>SHARE ROW EXCLUSIVE</b> protects a table against concurrent data changes, and is self-exclusive so that only one session can hold it at a time.<br><br>No SQL statements automatically acquire this lock mode.                                                                                                                                                                                                                             | -                                                                                                          |
| EXCLUSIVE           | <b>EXCLUSIVE</b> allows concurrent queries of the target table but does not allow any other operations.<br><br>This mode allows only concurrent <b>ACCESS SHARE</b> locks; that is, only reads from the table can proceed in parallel with a transaction holding this lock mode.<br><br>No SQL statements automatically acquire this lock mode on user tables. However, it will be acquired on some system tables in case of some operations. | -                                                                                                          |
| ACCESS EXCLUSIVE    | This mode guarantees that the holder is the only transaction accessing the table in any way.<br><br>Acquired by the <b>ALTER TABLE</b> , <b>DROP TABLE</b> , <b>TRUNCATE</b> , <b>REINDEX</b> , <b>CLUSTER</b> , and <b>VACUUM FULL</b> commands.                                                                                                                                                                                             | This is also the default lock mode for <b>LOCK TABLE</b> statements that do not specify a mode explicitly. |
| UPDATE EXCLUSIVE    | The <b>UPDATE EXCLUSIVE</b> lock allows concurrent <b>(AUTO) VACUUM</b> and <b>(AUTO) ANALYZE</b> , but does not allow concurrent <b>(AUTO) VACUUM</b> .<br><b>NOTE</b> <ul style="list-style-type: none"><li>This parameter is supported only by clusters of version 8.2.1.200 or later.</li><li>The <b>UPDATE EXCLUSIVE</b> lock is used only in the <b>VACUUM</b> syntax.</li></ul>                                                        | -                                                                                                          |

| Parameter              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range |
|------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| NOWAIT                 | Specifies that <b>LOCK TABLE</b> should not wait for any conflicting locks to be released: if the specified locks cannot be acquired immediately without waiting, the transaction is aborted.<br><br>If <b>NOWAIT</b> is not specified, <b>LOCK TABLE</b> obtains a table-level lock, waiting if necessary for any conflicting locks to be released.                                                                                                                                              | -           |
| LOCAL COORDINATOR ONLY | Specifies that <b>LOCK TABLE</b> is executed only on the CN that receives the current session request and is not delivered to other CNs or all DN. This is used only for metadata operations to improve efficiency.<br><br><b>NOTICE</b> <ul style="list-style-type: none"><li>• This parameter is supported by version 8.2.0.100 or later clusters.</li><li>• Currently, only the <b>ACCESS SHARE</b> lock mode is supported. If other lock modes are used, an error will be reported.</li></ul> | -           |

## Examples

Obtain a **SHARE** lock on a primary key table when going to perform inserts into a foreign key table.

```
START TRANSACTION;

LOCK TABLE tpceds.reason IN SHARE MODE;

SELECT r_reason_desc FROM tpceds.reason WHERE r_reason_sk=5;
r_reason_desc

Parts missing
(1 row)

COMMIT;
```

Obtain a **SHARE ROW EXCLUSIVE** lock on a primary key table when performing a delete operation.

```
CREATE TABLE tpceds.reason_t1 AS TABLE tpceds.reason;

START TRANSACTION;

LOCK TABLE tpceds.reason_t1 IN SHARE ROW EXCLUSIVE MODE;

DELETE FROM tpceds.reason_t1 WHERE r_reason_desc IN(SELECT r_reason_desc FROM tpceds.reason_t1
WHERE r_reason_sk < 6);
```

```
DELETE FROM tpcds.reason_t1 WHERE r_reason_sk = 7;
COMMIT;
```

Delete the **tpcds.reason\_t1** table.

```
DROP TABLE tpcds.reason_t1;
```

Add the ACCESS SHARE lock to the table and set the lock scope to the current CN.

```
BEGIN;
LOCK TABLE lock_test IN ACCESS SHARE MODE LOCAL COORDINATOR ONLY;
SELECT pg_get_tabledef('lock_test');
END;
```

## 21.14.8 MERGE INTO

### Function

**MERGE INTO** matches data in a source table with that in a target table based on a join condition. If data matches, **UPDATE** will be executed on the target table. Otherwise, **INSERT** will be executed. **MERGE INTO** handles both UPDATE and INSERT operations within a single statement, potentially improving performance.

### Precautions

- To run **MERGE INTO**, you must have the UPDATE and INSERT permissions for the target table, as well as the SELECT permission for the source table.
- **PREPARE** is not supported.
- **MERGE INTO** cannot be executed during redistribution.
- **MERGE INTO** cannot be executed for target tables that contain triggers.
- When executing **MERGE INTO** for the round-robin table, you are advised to disable the GUC parameter **allow\_concurrent\_tuple\_update**, or some **MERGE INTO** statements are not supported.

### Syntax

```
MERGE [/*+ plan_hint */] INTO table_name [partition_clause] [[AS] alias]
USING { { table_name | view_name } | subquery } [partition_clause] [[AS] alias]
ON (condition)
[
 WHEN MATCHED THEN
 UPDATE SET { column_name = { expression | DEFAULT } |
 (column_name [, ...]) = ({ expression | DEFAULT } [, ...]) } [, ...]
 [WHERE condition]
]
[
 WHEN NOT MATCHED THEN
 INSERT { DEFAULT VALUES |
 [(column_name [, ...])] VALUES ({ expression | DEFAULT } [, ...]) [, ...] [WHERE condition] }
];
```

where partition\_clause can be:

```
PARTITION (partition_name) | PARTITION FOR (partition_key_value [, ...])
```

## Parameter Description

Table 21-289 MERGE INTO parameters

| Parameter                  | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Value Range       |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| <b>plan_hint</b><br>clause | Following the keyword in the <i>/*+ */</i> format, hints are used to optimize the plan generated by a specified statement block. For details, see "DWS Performance Tuning" > "SQL Tuning" > "Hint-based Tuning" in <i>Data Warehouse Service (DWS) Developer Guide</i> .                                                                                                                                                                                                                                                                                                                                                                                  | -                 |
| <b>INTO</b> clause         | <p>Specifies the target table that is being updated or has data being inserted.</p> <ul style="list-style-type: none"><li>• <b>table_name</b><br/>Specifies the name of the target table.</li><li>• <b>partition_clause</b><br/>Specifies a partition in the target table. This parameter is supported only by clusters of version 9.1.0 or later.<br/><br/>Partitions can be specified only if the target table is a partitioned one. The specified partition name must already exist in the target table or the partition specified by the partition key already exists.</li><li>• <b>alias</b><br/>Specifies the alias for the target table.</li></ul> | Valid table name. |

| Parameter           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range              |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| <b>USING</b> clause | <p>Specifies the source table, which can be a table, view, or subquery.</p> <ul style="list-style-type: none"><li>• <b>partition_clause</b><br/>Specifies a partition in the source table. This parameter is supported only by clusters of version 9.1.0 or later.<br/><br/>Partitions can be specified only if the source table is a partitioned one. Partitions cannot be specified for views or subqueries. The specified partition name must already exist in the source table or the partition specified by the partition key already exists.</li><li>• <b>alias</b><br/>Specifies the alias for the target table.</li></ul> | Valid source table name. |
| <b>ON</b> clause    | <p>Specifies the condition used to match data between the source and target tables.</p> <ul style="list-style-type: none"><li>• Columns in the condition cannot be updated.</li><li>• The <b>ON</b> association condition can be <b>ctid</b>, <b>xc_node_id</b>, or <b>tableoid</b>.</li></ul>                                                                                                                                                                                                                                                                                                                                    | -                        |

| Parameter                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Value Range |
|-------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| <b>WHEN MATCHED</b><br>clause | <p>Performs <b>UPDATE</b> if data in the source table matches that in the target table based on the condition.</p> <ul style="list-style-type: none"><li>• Distribution keys cannot be updated.</li><li>• System catalogs and system columns cannot be updated.</li><li>• With a partition specified in the target table, if the <b>UPDATE</b> causes the partition keys to change:<ul style="list-style-type: none"><li>– If the target table is a column-store table, the error message "The inserted partition key is not mapped to the specified 'partition_name' partition" is displayed.</li><li>– If the target table is a row-store table:<br/>If <b>ENABLE ROW MOVEMENT</b> is set for the target table, no error will be reported, the updated data will belong to the new partition.<br/>If <b>Disable ROW MOVEMENT</b> is set for the target table, the error message "fail to update partitioned table 'table_name'" will be displayed.</li></ul></li></ul> | -           |



| Parameter                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Value Range                                                                     |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <b>WHEN NOT MATCHED</b> clause | <p>Specifies that the <b>INSERT</b> operation is performed if data in the source table does not match that in the target table based on the condition.</p> <ul style="list-style-type: none"><li>An <b>INSERT</b> clause cannot contain multiple <b>VALUES</b>.</li><li>The order of <b>WHEN MATCHED</b> and <b>WHEN NOT MATCHED</b> clauses can be reversed. One of them can be used by default, but they cannot be both used at one time. Two <b>WHEN MATCHED</b> or <b>WHEN NOT MATCHED</b> clauses cannot be specified at the same time.</li><li>If a partition in the target table is specified and more data beyond the scope of the partition is inserted, the error message "The inserted partition key is not mapped to the specified 'partition_name' partition" will be displayed.</li></ul> | -                                                                               |
| DEFAULT                        | Sets the column to its default value.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   | The value is <b>NULL</b> if no specified default value has been assigned to it. |
| WHERE condition                | <p>Specifies the conditions for the <b>UPDATE</b> and <b>INSERT</b> clauses. The two clauses will be executed only when the conditions are met. This parameter can be left empty.</p> <p>System columns cannot be referenced in <b>WHERE condition</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | -                                                                               |

## Examples

Create the target table **products** and source table **newproducts**, and insert data to them.

```
CREATE TABLE products
(
 product_id INTEGER,
 product_name VARCHAR2(60),
 category VARCHAR2(60)
);

INSERT INTO products VALUES (1501, 'vivitar 35mm', 'electronics');
INSERT INTO products VALUES (1502, 'olympus is50', 'electronics');
INSERT INTO products VALUES (1600, 'play gym', 'toys');
INSERT INTO products VALUES (1601, 'lamaze', 'toys');
```

```
INSERT INTO products VALUES (1666, 'harry potter', 'dvd');

CREATE TABLE newproducts
(
 product_id INTEGER,
 product_name VARCHAR2(60),
 category VARCHAR2(60)
);

INSERT INTO newproducts VALUES (1502, 'olympus camera', 'electronics');
INSERT INTO newproducts VALUES (1601, 'lamaze', 'toys');
INSERT INTO newproducts VALUES (1666, 'harry potter', 'toys');
INSERT INTO newproducts VALUES (1700, 'wait interface', 'books');
```

Run **MERGE INTO**. If the **product\_id** field is matched and the **product\_name** field in the target table is not **play gym**, UPDATE is executed. If the **product\_id** field is not matched and the **category** field in the source table is **books**, INSERT is executed.

```
MERGE INTO products p
USING newproducts np
ON (p.product_id = np.product_id)
WHEN MATCHED THEN
 UPDATE SET p.product_name = np.product_name, p.category = np.category WHERE p.product_name !=
'play gym'
WHEN NOT MATCHED THEN
 INSERT VALUES (np.product_id, np.product_name, np.category) WHERE np.category = 'books';
```

View the query result. The data in **product\_id** of rows 1502 and 1666 in the **products** table was updated, and the data in row 1700 was newly inserted.

```
SELECT * FROM products ORDER BY product_id;
product_id | product_name | category
-----+-----+-----
1501 | vivitar 35mm | electronics
1502 | olympus camera | electronics
1600 | play gym | toys
1601 | lamaze | toys
1666 | harry potter | toys
1700 | wait interface | books
(6 rows)
```

## 21.14.9 INSERT and UPSERT

### 21.14.9.1 INSERT

#### Function

**INSERT** inserts new rows into a table.

#### Precautions

- For details about how to use INSERT, see the "INSERT Operation" in *Data Warehouse Service (DWS) Developer Guide*.
- You must have the **INSERT** permission on a table in order to insert into it.
- Use of the **RETURNING** clause requires the **SELECT** permission on all columns mentioned in **RETURNING**.
- If you use the **query** clause to insert rows from a query, you of course need to have the **SELECT** permission on any table or column used in the query.

- Using the **OVERWRITE** clause requires both **SELECT** and **TRUNCATE** permissions on the table since it deletes existing data before inserting new records.
- When you connect to a database compatible to Teradata and **td\_compatible\_truncation** is **on**, a long character string will be automatically truncated. If later **INSERT** statements (not involving foreign tables) insert long strings to columns of char- and varchar-typed columns in the target table, the system will truncate the long strings to ensure no strings exceed the maximum length defined in the target table.

#### NOTICE

If inserting multi-byte character data (such as Chinese characters) to database with the character set byte encoding (SQL\_ASCII, LATIN1), and the character data crosses the truncation position, the string is truncated based on its bytes instead of characters. Unexpected result will occur in tail after the truncation. If you want correct truncation result, you are advised to adopt encoding set such as UTF-8, which has no character data crossing the truncation position.

## Syntax

```
[WITH [RECURSIVE] with_query [, ...]]
INSERT [/*+ plan_hint */] [IGNORE | OVERWRITE] INTO table_name [partition_clause] [AS alias]
[(column_name [, ...])]
 { DEFAULT VALUES
 | VALUES { ({ expression | DEFAULT } [, ...]) } [, ...]
 | query }
 [ON DUPLICATE KEY duplicate_action | ON CONFLICT [conflict_target] conflict_action]
 [RETURNING { * | { output_expression [[AS] output_name] }, ... }] ;
```

where partition\_clause can be:

```
PARTITION (partition_name)
| PARTITION FOR (partition_key_value [, ...])
```

and duplicate\_action can be:

```
UPDATE { column_name = { expression | DEFAULT } |
 (column_name [, ...]) = ({ expression | DEFAULT } [, ...])
 } [, ...]
```

and conflict\_target can be one of:

```
({ index_column_name | (index_expression) } [COLLATE collation] [opclass] [, ...]) [WHERE
index_predicate]
ON CONSTRAINT constraint_name
```

and conflict\_action is one of:

```
DO NOTHING
DO UPDATE SET { column_name = { expression | DEFAULT } |
 (column_name [, ...]) = ({ expression | DEFAULT } [, ...])
 } [, ...]
[WHERE condition]
```

## Parameter Description

**Table 21-290** INSERT parameters

| Parameter                                   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Value Range or Example                                                                      |
|---------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| WITH<br>[ RECURSIVE ]<br>with_query [, ...] | <p>The <b>WITH</b> clause allows you to specify one or more subqueries that can be referenced by name in the primary query, equal to temporary table.</p> <p>If <b>RECURSIVE</b> is specified, it allows a <b>SELECT</b> subquery to reference itself by name.</p> <p>The format of <b>with_query</b> is as follows:<br/>with_query_name [ ( column_name [, ...] ) ] AS<br/>( {select   values   insert   update   delete} )</p> <p><b>with_query_name</b> specifies the name of the result set generated by a subquery. Such names can be used to access the result sets of subqueries in a query.</p> <p><b>column_name</b> specifies the column name displayed in the subquery result set.</p> <p>Each subquery can be a <b>SELECT</b>, <b>VALUES</b>, <b>INSERT</b>, <b>UPDATE</b> or <b>DELETE</b> statement.</p> | <a href="#">Generating Temporary Tables with WITH and Inserting Data into Another Table</a> |
| <b>plan_hint</b><br>clause                  | Following the keyword in the <b>/*+ */</b> format, hints are used to optimize the plan generated by a specified statement block. For details, see "DWS Performance Tuning" > "SQL Tuning" > "Hint-based Tuning" in <i>Data Warehouse Service (DWS) Developer Guide</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | -                                                                                           |
| IGNORE                                      | <p>Specifies that the data that duplicates an existing primary key or unique key value will be ignored.</p> <p>For details, see <a href="#">UPSERT</a>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | <a href="#">Using IGNORE to Ignore Primary Key Conflicts</a>                                |

| Parameter | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Value Range or Example |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| OVERWRITE | <p>Specifies the overwrite mode. After this mode is used, the original data is cleared and only the newly inserted data exists. This parameter is available only for clusters of version 8.1.1 or later.</p> <p>You can specify the columns on which <b>OVERWRITE</b> takes effect, and the other columns will keep their original data. If a column has no original data, its value is <b>NULL</b>.</p> <p><b>INSERT OVERWRITE</b> constraint:</p> <ul style="list-style-type: none"><li>• Do not perform <b>OVERWRITE</b> and <b>INSERT INTO</b> operations at the same time. Otherwise, data written in real time may be unexpectedly cleared.</li><li>• <b>INSERT OVERWRITE</b> is unavailable for replication tables. <b>OVERWRITE</b> applies to the scenario where a large amount of data is imported. You are not advised to use <b>OVERWRITE</b> to insert a small amount of data.</li><li>• Do not concurrently perform insert overwrite operations on the same table. Otherwise, an error similar to "tuple concurrently updated." will be reported.</li><li>• If the cluster is being scaled out and data redistribution is required for the table where <b>INSERT OVERWRITE</b> is performed, <b>INSERT OVERWRITE</b> clears the current data and automatically distributes the inserted data to the new nodes after scale-out. If <b>INSERT OVERWRITE</b> and the data redistribution of the table are performed at the same time, <b>INSERT OVERWRITE</b> will interrupt the data redistribution of the table.</li><li>• <b>OVERWRITE</b> allows you to insert data into an OBS foreign table in overwrite mode, where the data in the original directory will be overwritten.</li><li>• The foreign table supports <b>OVERWRITE</b> single-partition and multi-level Value partitioning. This function is supported only in 9.1.0.100 and later versions.</li></ul> | -                      |

| Parameter      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Value Range or Example         |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
|                | <p>Example: INSERT OVERWRITE into table_name partition for (p1_value, p2_value...) ...</p> <ul style="list-style-type: none"><li>When <b>INSERT OVERWRITE</b> and <b>ALTER TABLE...ADD PARTITION</b> are both executed on a given table, the number of partitions on different nodes may be inconsistent. As a result, an error is reported during the execution of <b>INSERT OVERWRITE</b>. You are advised to run <b>INSERT OVERWRITE</b> again.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                |
| table_name     | Specifies the name of the target table.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | An existing table name.        |
| AS             | Specifies an alias for the target table <i>table_name</i> . <i>alias</i> indicates the alias name.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | -                              |
| column_name    | <p>Specifies the name of a column in a table.</p> <ul style="list-style-type: none"><li>The column name can be qualified with a subfield name or array subscript, if needed.</li><li>Each column not present in the explicit or implicit column list will be filled with a default value, either its declared default value or <b>NULL</b> if there is none. (Inserting into only some fields of a composite column leaves the other fields null.)</li><li>The target column names <b>column_name</b> can be listed in any order. If no list of column names is given at all, the default is all the columns of the table in their declared order.</li><li>The target columns are the first N column names, if there are only N columns supplied by the value clause or query.</li><li>The values supplied by the <b>value</b> clause or <b>query</b> are associated with the explicit or implicit column list left-to-right.</li></ul> | An existing column name.       |
| partition_name | Specifies the name of a partition.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Name of an existing partition. |

| Parameter           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                         | Value Range or Example                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| partition_key_value | Specifies the key value of a partition.<br>The value specified by <b>PARTITION FOR ( partition_key_value [, ...] )</b> can uniquely identify a partition. Multiple partition key values can be specified. Use commas (,) to separate multiple partition key values.                                                                                                                                                                                 | Value range of the partition key of the partition to be renamed.                                                                                                         |
| expression          | Specifies an expression or a value to assign to the corresponding column. <ul style="list-style-type: none"><li>• If single-quotation marks are inserted in a column, the single-quotation marks need to be used for escape.</li><li>• If the expression for any column is not of the correct data type, automatic type conversion will be attempted. If the attempt fails, data insertion fails and the system returns an error message.</li></ul> | <a href="#">Inserting an Expression or Value into a Column</a>                                                                                                           |
| DEFAULT             | Specifies the default value of the corresponding field name.                                                                                                                                                                                                                                                                                                                                                                                        | The value is <b>NULL</b> if no specified default value has been assigned to it. For details, see <a href="#">Using the Default Value of a Column When Writing Data</a> . |
| query               | Specifies a query statement ( <b>SELECT</b> statement) that uses the query result as the inserted data.                                                                                                                                                                                                                                                                                                                                             | <a href="#">Inserting Query Results into a Table</a>                                                                                                                     |
| ON DUPLICATE KEY    | Specifies that the data that duplicates an existing primary key or unique key value will be updated.<br><i>duplicate_action</i> specifies the columns and data to be updated.<br>For details, see <a href="#">UPSERT</a> .                                                                                                                                                                                                                          | <a href="#">Inserting Data with ON DUPLICATE KEY for Handling Primary Key/Unique Conflicts</a>                                                                           |

| Parameter         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Value Range or Example                                                                                                   |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| ON CONFLICT       | <p>Specifies that the data that duplicates an existing primary key or unique key value will be ignored or updated.</p> <p><i>conflict_target</i> specifies the column name <i>index_column_name</i>, expression <i>index_expression</i> that contains multiple column names, or constraint name <i>constraint_name</i>. It is used to infer whether there is a unique index from the column name, the expression that contains multiple column names, or the constraint name. <i>index_column_name</i> and <i>index_expression</i> must comply with the index column format of <b>CREATE INDEX</b>.</p> <p><i>conflict_action</i> specifies the policy to be executed upon a primary key or unique constraint conflict. There are two available actions:</p> <ul style="list-style-type: none"><li>• <b>DO NOTHING</b>: Ignore the conflict.</li><li>• <b>DO UPDATE SET</b>: Update data upon a conflict. The columns and data to be updated must be specified.</li></ul> <p>For details, see <a href="#">UPSERT</a>.</p> | <a href="#">Examples</a>                                                                                                 |
| RETURNING         | Returns the inserted rows. The syntax of the <b>RETURNING</b> list is identical to that of the output list of <b>SELECT</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <a href="#">Using RETURNING to Return Insertion Results</a>                                                              |
| output_expression | An expression used to calculate the output of the <b>INSERT</b> command after each row is inserted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | The expression can use any column names of the table. Write * to return all columns of the inserted row(s).              |
| output_name       | Specifies a name to use for a returned column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |



## Inserting One or More Data Records

Create the **reason\_t1** table.

```
DROP TABLE IF EXISTS reason_t1;
CREATE TABLE reason_t1
(
 TABLE_SK INTEGER ,
 TABLE_ID VARCHAR(20) ,
 TABLE_NA VARCHAR(20)
);
```

Insert one data record to the table.

```
INSERT INTO reason_t1(TABLE_SK, TABLE_ID, TABLE_NA) VALUES (1, 'S01', 'StudentA');
```

Insert one data record into the table. This command is equivalent to the last one.

```
INSERT INTO reason_t1 VALUES (1, 'S01', 'StudentA');
```

Insert records whose **TABLE\_SK** is less than **1** into the table.

```
INSERT INTO reason_t1 SELECT * FROM reason_t1 WHERE TABLE_SK < 1;
```

Insert records into the table.

```
INSERT INTO reason_t1 VALUES (1, 'S01', 'StudentA'),(2, 'T01', 'TeacherA'),(3, 'T02', 'TeacherB');
SELECT * FROM reason_t1 ORDER BY 1;
```

| table_sk | table_id | table_na |
|----------|----------|----------|
| 1        | S01      | StudentA |
| 1        | S01      | StudentA |
| 1        | S01      | StudentA |
| 2        | T01      | TeacherA |
| 3        | T02      | TeacherB |

## Inserting Query Results into a Table

Insert data back into the **reason\_t1** table.

```
INSERT INTO reason_t1 SELECT * FROM reason_t1;
```

## Using RETURNING to Return Insertion Results

Insert table data and return the values of **table\_id** and **table\_na** corresponding to the inserted data.

```
INSERT INTO reason_t1 SELECT * FROM reason_t1 RETURNING table_id, table_na;
```

| table_id | table_na |
|----------|----------|
| T01      | TeacherA |
| T02      | TeacherB |
| S01      | StudentA |
| T02      | TeacherB |
| T01      | TeacherA |

Insert data and return all column values of the inserted data.

```
INSERT INTO reason_t1 SELECT * FROM reason_t1 RETURNING *;
```

| table_sk | table_id | table_na |
|----------|----------|----------|
| 2        | T01      | TeacherA |
| 3        | T02      | TeacherB |
| 2        | T01      | TeacherA |
| 3        | T02      | TeacherB |
| 1        | S01      | StudentA |

Insert data and return the value of the renamed column **my\_id**.

```
INSERT INTO reason_t1 VALUES (1, 'S01', 'StudentA') RETURNING table_id AS my_id;
```

| my_id |
|-------|
| S01   |

## Using INSERT OVERWRITE to Insert Data in Overwriting Mode

Use INSERT OVERWRITE to update data in a table, that is, insert data in overwriting mode (clear the data in the target table first).

```
INSERT OVERWRITE INTO reason_t1 values (5, 'S02', 'StudentB');
SELECT * FROM reason_t1 ORDER BY 1;
```

| table_sk | table_id | table_na |
|----------|----------|----------|
| 5        | S02      | StudentB |

## Using the Default Value of a Column When Writing Data

```
DROP TABLE IF EXISTS products;
CREATE TABLE products (
```

```
id INT PRIMARY KEY,
name VARCHAR(50),
price DECIMAL(10, 2) DEFAULT 0.00
);
```

Insert a data record. If the value of the **price** column is not defined, **0.00** is used by default.

```
INSERT INTO products (id, name)VALUES (201, 'phone'); -- The price column uses the default value 0.00.
```

Insert a data record and explicitly use the default value.

```
INSERT INTO products (id, name, price)VALUES (202, 'pad', DEFAULT); -- Explicitly use the default value.
SELECT * FROM product;
```

| id  | name  | price |
|-----|-------|-------|
| 201 | phone | 0.00  |
| 202 | pad   | 0.00  |

## Generating Temporary Tables with WITH and Inserting Data into Another Table

Insert some data in a table to another table: Use the **WITH** subquery to obtain a temporary table **temp\_t**, and then insert all data in **temp\_t** to another table **reason\_t1**.

```
WITH temp_t AS (SELECT * FROM reason_t1) INSERT INTO reason_t1 SELECT * FROM temp_t ORDER BY 1;
```

## Using IGNORE to Ignore Primary Key Conflicts

```
DROP TABLE IF EXISTS users;
CREATE TABLE users (
id INT PRIMARY KEY,
username VARCHAR(50)
)
DISTRIBUTE BY HASH (id);
```

Insert two records whose primary keys are both **1**. A conflict occurs and the operation cannot be performed.

```
INSERT INTO users (id, username)
VALUES (1, 'john_doe'),
(2, 'jane_smith'),
(1, 'duplicate_user'); --Primary key conflict. Insertion failed.
```

```
error_code: DWS.S0010010
error_msg: Constraint Violation. STATE: 23505, message: ERROR: dn_6005_6006: duplicate key value violates
unique constraint "users_pkey"
Detail: Key (id)=(1) already exists.
```

Use **IGNORE** to ignore the conflict. The conflicting data is ignored and skipped.

```
INSERT IGNORE INTO users (id, username)
VALUES (1, 'john_doe'),
(2, 'jane_smith'),
(1, 'duplicate_user'); -- Conflicting data will be ignored.
SELECT * FROM users;
```

| id | username   |
|----|------------|
| 1  | john_doe   |
| 2  | jane_smith |

## Inserting Data with ON DUPLICATE KEY for Handling Primary Key/Unique Conflicts

```
DROP TABLE IF EXISTS user_logins;
CREATE TABLE user_logins
(user_id int primary key,
login_count int)
DISTRIBUTE BY HASH (user_id);
```

Insert two records. The primary key of the second record conflicts with that of the first record. The insertion fails.

```
INSERT INTO user_logins VALUES (1, 1);
INSERT INTO user_logins VALUES (1, 1); --Primary key conflict. Insertion failed.
SELECT * FROM user_logins;
```

| user_id | login_count |
|---------|-------------|
| 1       | 1           |

If a duplicate key exists, increment the **login\_count** field by 1 using the **ON DUPLICATE KEY** clause.

```
INSERT INTO user_logins VALUES (1, 1) ON DUPLICATE KEY UPDATE login_count = login_count + 1;
SELECT * FROM user_logins;
```

| user_id | login_count |
|---------|-------------|
| 1       | 2           |

## Inserting an Expression or Value into a Column

```
CREATE TABLE tt01 (id int,content varchar(50));
```

```
INSERT INTO tt01 values (1,'Jack say "hello"'); --Insert "Jack say'hello'" and add single quotation marks to escape.
```

```
INSERT INTO tt01 values (2,'Rose do 50%');
```

```
INSERT INTO tt01 values (3,'Lilei say "world"'); --Insert "Lilei say'world'" and add single quotation marks to escape.
```

```
INSERT INTO tt01 values (4,'Hanmei do 100%');
```

```
SELECT * FROM tt01;
```

| id | content           |
|----|-------------------|
| 3  | Lilei say 'world' |
| 2  | Rose do 50%       |
| 1  | Jack say 'hello'  |
| 4  | Hanmei do 100%    |

## Inserting Data into a Specified Partition of a Partitioned Table

Insert data into a specified partition of a partitioned table.

```
CREATE TABLE test_range_row(a int, d int)
DISTRIBUTE BY hash(a) PARTITION BY RANGE(d)
(
 PARTITION p1 values LESS THAN (60),
 PARTITION p2 values LESS THAN (75),
 PARTITION p3 values LESS THAN (90),
 PARTITION p4 VALUES LESS THAN (maxvalue)
);
INSERT OVERWRITE INTO test_range_row PARTITION(p1) VALUES(55,51);
INSERT OVERWRITE INTO test_range_row PARTITION(p3) VALUES(85,80);

DELETE FROM test_range_row PARTITION(p1);
```

Insert data into a partition of a partitioned table.

```
CREATE FOREIGN TABLE test_ft(c1 text, c2 text, c3 text, c4 text)
PARTITION BY (c1,c2,c3) AUTOMAPPED;
```

Insert data into the level-1 (p1\_1) partition.

```
INSERT OVERWRITE INTO test_ft PARTITION FOR(p1_1) VALUES(p1_1,2,3,4);
```

Insert data into the level-2 (p1\_2, p2\_2) partition.

```
INSERT OVERWRITE INTO test_ft PARTITION FOR(p1_2,p2_2) VALUES(p1_2,p2_2,3,4);
```

Insert data into the level-3 (p1\_3,p2\_3,p3\_3) partition.

```
INSERT OVERWRITE INTO test_ft PARTITION FOR(p1_3,p2_3,p3_3) VALUES(p1_3,p2_3,p3_3,4);
```

## Setting hive\_compat\_options to Avoid Interpreting Empty Strings as 0 in MySQL Compatibility Mode

In MySQL compatibility mode, DWS treats an empty string as 0 during data operations by default.

For example, importing data from Hive to DWS in finance can cause issues if an empty string is treated as 0. Since these values mean different things in finance, this mistake alters the data, reducing its accuracy and trustworthiness.

To ensure data integrity and consistency during data import, you can set the GUC parameter `hive_compat_options` to **empty\_str\_to\_null** for conversion logic. When an empty string is inserted, the system converts the empty string to NULL.

Create a MySQL-compatible database.

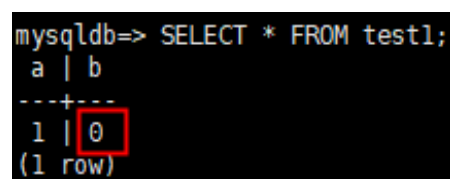
```
CREATE DATABASE mysqlpdb DBCOMPATIBILITY 'MYSQL';
```

Reconnect to the new database **mysqlpdb**, run the following statements, and insert an empty string.

```
CREATE TABLE test1 (a int, b int);
INSERT INTO test1 VALUES (1, "");
```

Query the data. The empty string is set to 0.

```
SELECT * FROM test1;
```



```
mysqlpdb=> SELECT * FROM test1;
a | b
---+---
1 | 0
(1 row)
```

Set the GUC parameter **hive\_compat\_options** to **empty\_str\_to\_null**.

```
ALTER DATABASE mysqlpdb SET hive_compat_options TO empty_str_to_null;
```

Insert data again.

```
INSERT INTO test1 VALUES (2, "");
```

Query the data. The empty string is automatically converted to NULL.

```
SELECT * FROM test1;
```

```
mysqlpdb=> SELECT * FROM test1;
a | b
---+---
1 | 0
2 |
(2 rows)
```

## Optimization

- VALUES

When you insert data in batches using the **INSERT** statement, you are advised to combine multiple records into one statement to insert data, improving data loading performance. Example: **INSERT INTO sections VALUES (30, 'Administration', 31, 1900), (40, 'Development', 35, 2000), (50, 'Development', 60, 2001);**

### 21.14.9.2 UPSERT

## Function

**UPSERT** inserts rows into a table. When a primary key or unique constraint conflict occurs, it can choose to update the conflicting data or ignore the conflict.

---

#### NOTICE

The **UPSERT** syntax is supported only in 8.1.1 and later.

---

## Precautions

---

#### WARNING

- Do not concurrently update the same column-store table using the **UPDATE** and **UPSERT** statements.
  - For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Developer Guide*.
- 
- Permission requirements: Only users with the **INSERT** or **UPDATE** permission on a table can run the **UPSERT** statement to insert data to or update data in the table.

- Index requirements: The **UPSERT** statement of updating data upon conflict can be executed only when the target table contains a primary key or unique index. The **UPSERT** statement of updating data upon conflict cannot be executed if no unique indexes are available. You can execute the statement only after the indexes are rebuilt.
- Other requirements:
  - The **WITH** clause cannot be used at the same time.
  - **INSERT OVERWRITE** cannot be used at the same time.
  - The **UPDATE** clause and its **WHERE** clause do not support subqueries.
  - **VALUES(colname)** in the **UPDATE** clause does not support outer nested functions. That is, the usage similar to **sqrt(VALUES(colname))** is not supported. To support this function, use the **EXCLUDED.colname** syntax.
  - **INSERT INTO ON CONFLICT(...) DO UPDATE** must contain **conflict\_target**. That is, a column or constraint name must be specified.
  - The distribution column cannot be updated. (Exception: Update is allowed if the distribution key is the same as the updated value.)

```
CREATE TABLE t1(dist_key int PRIMARY KEY, a int, b int);
INSERT INTO t1 VALUES(1,2,3) ON CONFLICT(dist_key) DO UPDATE SET dist_key =
EXCLUDED.dist_key, a = EXCLUDED.a + 1;
INSERT INTO t1 VALUES(1,2,3) ON CONFLICT(dist_key) DO UPDATE SET dist_key = dist_key, a =
EXCLUDED.a + 1;
```
  - The **UPSERT** statement cannot be executed on the target table that contains a trigger (with the **INSERT** or **UPDATE** trigger event).
  - The **UPSERT** statement is not supported for updatable views.
  - The **UPDATE** clause, the **WHERE** clause of **UPDATE**, and the index condition expression should not contain **functions that cannot be pushed down**.
  - Unique indexes cannot be deferred.
  - When performing the update operation of **UPSERT** using **INSERT INTO SELECT**, pay attention to the query result sequence of **SELECT**. In a distributed environment, if the **ORDER BY** statement is not used, the sequence of returned results may be different each time the same **SELECT** statement is executed. As a result, the execution result of the **UPSERT** statement does not meet the expectation.
  - Multiple updates are not supported. When conflicting data groups exist, you will see an error like "INSERT ON CONFLICT DO UPDATE command cannot affect row a second time." This does not happen if the query uses a PGXC plan. For details, see [FAQs](#).
- Impact on performance:
  - When **UPSERT** is executed on column-store tables, enable DELTA tables to avoid small CUs from being generated. A large number of small CUs may cause tables to bloat or lead to poor query performance.
  - **UPSERT**, **UPDATE**, and **DELETE** operations cannot be concurrently performed because they need to wait for the CU lock. This problem cannot be solved even if the DELTA table is enabled. To execute **UPSERT**, **UPDATE**, and **DELETE** operations concurrently on column-store tables, use HStore tables.
- Deadlock: A distributed deadlock may occur, resulting in query hanging. When many **UPSERT** commands run together in a transaction or batch via JDBC

with **setAutoCommit(false)**, deadlocks can happen if several tasks try to update the same row simultaneously. This occurs because different threads on various nodes might update rows in varying orders. To solve this issue, you can:

- a. Decrease the value of the GUC parameter **lockwait\_timeout** (default value: 20 minutes). A distributed deadlock error will be reported after waiting for *the value of lockwait\_timeout*. You can decrease the value of this parameter to reduce the service waiting time caused by a deadlock.
- b. Only import data with the same primary key through one database connection. You can run **UPSERT** statements at the same time.
- c. Ensure that only one UPSERT statement is executed in each transaction. In this way, **UPSERT** statements can be executed concurrently.
- d. Run all **UPSERT** operations one at a time in a single thread; avoid running them simultaneously.

Method 1 reduces waiting time but does not fix deadlocks. When using the **UPSERT** statement in services, lower this parameter's value. Methods 2, 3, and 4 resolve deadlocks, with method 2 being preferred due to its superior performance.

## Syntax

For details, see [Syntax](#) of **INSERT**. The following table describes the syntax of **UPSERT**.

The first syntax does not specify an index. The system checks for conflicts on all primary keys or unique indexes. If a conflict exists, the system ignores or updates the corresponding data.

The second syntax is to specify an index. The primary key or unique index is inferred from the column name, the expression that contains column names, or the constraint name specified in the **ON CONFLICT** clause.

- Scenario 1: If inserting data causes a conflict, the system updates the current data with the new information.
  - Conflict update without specifying an index:  
`INSERT INTO ON DUPLICATE KEY UPDATE`
  - Conflict update with an index specified:  
`INSERT INTO ON CONFLICT(...) DO UPDATE SET`  
`INSERT INTO ON CONFLICT ON CONSTRAINT con_name DO UPDATE SET`
- Scenario 2: If a data insertion causes a conflict, the system skips the operation.
  - Conflict ignorance without specifying an index:  
`INSERT IGNORE`  
`INSERT INTO ON CONFLICT DO NOTHING`
  - Conflict ignorance with an index specified:  
`INSERT INTO ON CONFLICT(...) DO NOTHING`  
`INSERT INTO ON CONFLICT ON CONSTRAINT con_name DO NOTHING`



Table 21-291 UPSERT parameters

| Parameter         | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range or Example |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| ON CONFLICT (...) | <p>The system determines the primary key or unique index by analyzing the given column or constraint names for a specific index's syntax. You can specify a single column name or multiple column names by using an expression, for example, <b>(column1, column2, column3)</b>.</p> <p><b>collation</b> and <b>opclass</b> can be specified when you create an index. Therefore, you can also specify them after the column name for index inference.</p> <p><b>COLLATE collation</b> specifies the collation of a column, and <b>opclass</b> specifies the name of the operator class. For details, see <a href="#">CREATE INDEX</a>.</p> <p>When inferring the unique index from an expression that includes multiple column names, the system checks whether there is a unique index that exactly contains all the column names specified by <b>conflict_target</b>.</p> <ul style="list-style-type: none"><li>• If <b>collation</b> and <b>opclass</b> are not specified, a match is considered found as long as a column has the same name as the specified single column or multiple columns have the same names as those specified by the column expression (regardless of the values of <b>collation</b> and <b>opclass</b> specified for the index column).</li><li>• If <b>collation</b> and <b>opclass</b> are specified, their values must also match the <b>collation</b> and <b>opclass</b> of the index.</li></ul> | -                      |
| UPDATE clause     | <p>The <b>UPDATE</b> clause can use <b>VALUES(colname)</b> or <b>EXCLUDED.colname</b> to reference inserted data. <b>EXCLUDED</b> indicates the data row that should be excluded due to conflict. For details about the syntax example, see <a href="#">Examples</a>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | -                      |

| Parameter           | Description                                                                                                                                                                                                                                                                                                                                                                                        | Value Range or Example |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <b>WHERE</b> clause | <ul style="list-style-type: none"><li>The <b>WHERE</b> clause is used to determine whether a specified condition is met when data conflict occurs. If yes, update the conflicting data. Otherwise, ignore the conflict.</li><li>Only syntax 2 of <b>Update Data Upon Conflict</b> can specify the <b>WHERE</b> clause, that is, <b>INSERT INTO ON CONFLICT(...) DO UPDATE SET WHERE</b>.</li></ul> | -                      |

## Examples

Create table **reason\_t2** and insert data into it.

```
DROP TABLE IF EXISTS reason_t2;
CREATE TABLE reason_t2
(
 a int primary key,
 b int,
 c int
);

INSERT INTO reason_t2 VALUES (1, 2, 3);
SELECT * FROM reason_t2 ORDER BY 1;
a | b | c
---+---+---
1 | 2 | 3
(1 rows)
```

Insert two data records into the table **reason\_t2**. One data record conflicts and the other does not. Conflicting data is ignored, and non-conflicting data is inserted.

```
INSERT INTO reason_t2 VALUES (1, 4, 5),(2, 6, 7) ON CONFLICT(a) DO NOTHING;
SELECT * FROM reason_t2 ORDER BY 1;
a | b | c
---+---+---
1 | 2 | 3
2 | 6 | 7
(2 rows)
```

Insert two data records into the table **reason\_t2**. One data record conflicts and the other does not. Update conflicting information and add new data where there is no conflict. **EXCLUDED.b** and **EXCLUDED.c** refer to columns **b** and **c** for the inserted values, which are 4 and 5 in this case.

```
INSERT INTO reason_t2 VALUES (1, 4, 5),(3, 8, 9) ON CONFLICT(a) DO UPDATE SET b = EXCLUDED.b, c = EXCLUDED.c;
SELECT * FROM reason_t2 ORDER BY 1;
a | b | c
---+---+---
1 | 4 | 5
2 | 6 | 7
3 | 8 | 9
(3 rows)
```

Filter the updated rows.

```
INSERT INTO reason_t2 VALUES (2, 7, 8) ON CONFLICT (a) DO UPDATE SET b = excluded.b, c = excluded.c
WHERE reason_t2.c = 7;
SELECT * FROM reason_t2 ORDER BY 1;
a | b | c
---+---+---
1 | 4 | 5
2 | 7 | 8
3 | 8 | 9
(3 rows)
```

Insert data into the table **reason\_t**. Update the conflicting data and adjust the mapping. That is, update column c to column b and column b to column c.

```
INSERT INTO reason_t2 VALUES (1, 2, 3) ON CONFLICT (a) DO UPDATE SET b = excluded.c, c = excluded.b;
SELECT * FROM reason_t2 ORDER BY 1;
a | b | c
---+---+---
1 | 3 | 2
2 | 7 | 8
3 | 8 | 9
(3 rows)
```

## FAQs

Multiple updates are not supported. When conflicting data groups exist, you will see an error like "INSERT ON CONFLICT DO UPDATE command cannot affect row a second time." This does not happen if the query uses a PGXC plan. Below is an example.

```
CREATE TABLE t1(id int PRIMARY KEY, a int, b int);
SET explain_perf_mode=pretty;
-- Use the stream query plan:
EXPLAIN (COSTS OFF) INSERT INTO t1 VALUES(1,2,3),(1,5,6) ON CONFLICT(id) DO UPDATE SET a =
EXCLUDED.a + 1;
 QUERY PLAN

id | operation
---+-----
1 | -> Streaming (type: GATHER)
2 | -> Insert on t1
3 | -> Streaming(type: REDISTRIBUTE)
4 | -> Values Scan on "VALUES*"
Predicate Information (identified by plan id)

2 --Insert on t1
 Conflict Resolution: UPDATE
 Conflict Arbiter Indexes: t1_pkey
===== Query Summary =====

System available mem: 819200KB
Query Max mem: 819200KB
Query estimated mem: 3104KB
(18 rows)
INSERT INTO t1 VALUES(1,2,3),(1,5,6) ON CONFLICT(id) DO UPDATE SET a = EXCLUDED.a + 1;
ERROR: dn_xxxx: INSERT ON CONFLICT DO UPDATE command cannot affect row a second time
HINT: Ensure that no rows proposed for insertion within the same command have duplicate constrained
values.
-- Disable the stream plan and generate a PGXC plan:
set enable_stream_operator = off;
EXPLAIN (COSTS OFF) INSERT INTO t1 VALUES(1,2,3),(1,5,6) ON CONFLICT(id) DO UPDATE SET a =
EXCLUDED.a + 1;
 QUERY PLAN

id | operation
---+-----
1 | -> Insert on t1
2 | -> Values Scan on "VALUES*"
Predicate Information (identified by plan id)
```

```

1 --Insert on t1
 Conflict Resolution: UPDATE
 Conflict Arbiter Indexes: t1_pkey
 Node expr: id
(11 rows)
INSERT INTO t1 VALUES(1,2,3),(1,5,6) ON CONFLICT(id) DO UPDATE SET a = EXCLUDED.a + 1;
INSERT 0 2
```

## 21.14.10 UPDATE

### Function

**UPDATE** updates data in a table. **UPDATE** changes the values of the specified columns in all rows that satisfy the condition. The **WHERE** clause clarifies conditions. The columns to be modified need be mentioned in the **SET** clause; columns not explicitly modified retain their previous values.

### Precautions

- For details about the UPDATE operation, see "UPDATE and DELETE Operations" in the *Data Warehouse Service (DWS) Developer Guide*.
- Permission requirements: To modify a table, you must have the UPDATE permission on the table and the SELECT permission on any table involved in the expression or condition.
- The distribution column of a table cannot be modified.
- You are not advised to create a table that needs to be frequently updated as a replication table.
- Constraints on column-store tables:
  - Column-store tables do not support the **RETURNING** clause.
  - Column-store tables do not support non-deterministic update. If you update data in one row with multiple rows of data in a column-store table, an error is reported.
  - Memory space that records update operations in column-store tables is not reclaimed. You need to clean it by executing **VACUUM FULL table\_name**.
  - Column-store tables support lightweight **UPDATE** operations, which only rewrite the updated columns to reduce space usage. To enable column-store lightweight **UPDATE**, configure **enable\_light\_colupdate**. It is disabled by default.
  - Column-store lightweight **UPDATE** is unavailable and automatically changes to the regular **UPDATE** operation in the following scenarios: updating an index column, updating a primary key column, updating a partition column, updating a PCK column, and online scaling.
  - There is a low probability that an error is reported when column-store lightweight **UPDATE** and backend column-store **AUTOVACUUM** coexist. To avoid such an issue, you can run **ALTER TABLE** to set the table-level parameter **enable\_column\_autovacuum\_garbage** to **off** (**off** means that the backend column-store **AUTOVACUUM** of tables is disabled).

### Syntax

```
UPDATE [/*+ plan_hint */] [ONLY] table_name [*] [[AS] alias]
[PARTITION (partition_name) | PARTITION FOR (partition_key_value [, ...])]
```

```
SET {column_name = { expression | DEFAULT }
| (column_name [, ...]) = { ({ expression | DEFAULT } [, ...]) |sub_query } }[, ...]
[FROM from_list] [WHERE condition]
[RETURNING {*
| {output_expression [[AS] output_name]} [, ...] }];
```

where sub\_query can be:

```
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
{ * | {expression [[AS] output_name]} [, ...] }
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]]
```

## Parameter Description

Table 21-292 UPDATE parameters

| Parameter                  | Description                                                                                                                                                                                                                                                                                         | Example                                                                                                                                                       |
|----------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>plan_hint</b><br>clause | Following the keyword in the <b>/#+ */</b> format, hints are used to optimize the plan generated by a specified statement block. For details, see "DWS Performance Tuning" > "SQL Tuning" > "Hint-based Tuning" in <i>Data Warehouse Service (DWS) Developer Guide</i> .                            | -                                                                                                                                                             |
| table_name                 | Specifies the name (optionally schema-qualified) of the table to be updated.                                                                                                                                                                                                                        | <b>test.table1</b><br>( <b>test</b> is the schema name).                                                                                                      |
| [AS] alias                 | Specifies a short alias for the table for easy reference. Its value is a string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                     | Update the <b>employees</b> table with alias <b>e</b> :<br>UPDATE employees<br>AS e SET e.salary =<br>e.salary * 1.1 WHERE<br>e.department =<br>'technology'; |
| partition_name             | Specifies the partition name. If this parameter is specified, the data corresponding to the partition name will be updated. It is available for range partitioned tables, list partitioned tables, and other named partitioned tables. Only clusters of version 8.2.1 or later support this option. | Update data in the partition <b>p1</b> :<br>UPDATE<br>test_range_row<br>PARTITION(p1) SET<br>d = 41;                                                          |
| partition_key_value        | Specifies the partition key value. If this parameter is specified, the data corresponding to the partition key value will be updated<br><br>The value specified by <b>PARTITION FOR (partition_key_value [, ...])</b> can uniquely identify a partition.                                            | Update the value of <b>b</b> to <b>20</b> for the partition with key 41:<br>UPDATE<br>test_range_row<br>PARTITION FOR (41)<br>SET b = 20;                     |

| Parameter   | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Example                                                                                                                                                                                 |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| column_name | <p>Renames a column.</p> <p>You can refer to this column by specifying the table name and column name of the target table. Example:<br/><code>UPDATE foo SET foo.col_name = 'GaussDB';</code></p> <p>You can refer to this column by specifying the target table alias and the column name. Example:<br/><code>UPDATE foo AS f SET f.col_name = 'GaussDB';</code></p>                                                                                                | -                                                                                                                                                                                       |
| expression  | An expression or value to assign to the column.                                                                                                                                                                                                                                                                                                                                                                                                                      | <p>Discount the product price by 20%:</p> <pre>UPDATE products SET price = price * 0.8 WHERE product_id = 101;</pre>                                                                    |
| DEFAULT     | Sets the column to its default value. The value is <b>NULL</b> if no specified default value has been assigned to it.                                                                                                                                                                                                                                                                                                                                                | <p>Reset the status and login times of inactive users:</p> <pre>UPDATE users SET status = DEFAULT, login_count = DEFAULT WHERE last_login &lt; CURRENT_DATE - INTERVAL '30 days';</pre> |
| sub_query   | <p>Specifies the subquery, which is used to update the target table based on the query results of other tables in the same database.</p> <p>For details about clauses in the <b>SELECT</b> statement, see <a href="#">SELECT</a>.</p>                                                                                                                                                                                                                                | <p>Update the average salary of a department using a subquery.</p> <pre>UPDATE departments d SET avg_salary = ( SELECT AVG(salary) FROM employees e WHERE e.dept_id = d.dept_id);</pre> |
| from_list   | <p>Specifies a table expression, that allows the columns from other tables to appear in the WHERE condition and update the target table. This is similar to the list of tables that can be specified in the <b>FROM</b> clause of a <b>SELECT</b> statement.</p> <p><b>NOTICE</b></p> <p>Note that the target table must not appear in the <b>from_list</b>, unless you intend a self-join (in which case it must appear with an alias in the <b>from_list</b>).</p> | <p><b>Example: Use the FROM from_list Clause to Preferentially Process Orders of Customers with High VIP Levels</b></p>                                                                 |

| Parameter         | Description                                                                                                                                                                                                                           | Example                                                                                                                                                                                                                                                                                  |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| condition         | Specifies an expression that follows WHERE and returns a result of the boolean type. Only rows for which this expression returns <b>true</b> are updated.                                                                             | Update the values of the students whose math scores are lower than 80:<br>UPDATE<br>student_scores SET<br>math_score =<br>math_score + 5,<br>update_time =<br>NOW() WHERE<br>math_score < 80;                                                                                            |
| output_expression | Specifies an expression to be computed and returned by the <b>UPDATE</b> command after each row is updated.<br><br>Value range: any table and table columns listed in FROM. The asterisk (*) indicates that all columns are returned. | Update and return the stock information before and after the update:<br>UPDATE inventory<br>SET stock = stock -<br>10 WHERE category<br>= 'Electronics'<br>RETURNING<br>item_id AS "Product<br>ID", name AS<br>"Product name",<br>stock AS "New<br>stock", stock + 10<br>AS "Old stock"; |
| output_name       | Specifies the name of the column returned by RETURNING to optimize the output. Its value is a string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                  | -                                                                                                                                                                                                                                                                                        |

## Examples: Common UPDATE Scenarios

Create a sample table and insert data into it.

```
DROP TABLE IF EXISTS products;
CREATE TABLE products (
 product_id INT PRIMARY KEY, --Product ID
 name VARCHAR(100), --Product name
 price DECIMAL(10,2), --Product price
 stock INT, --Product stock
 last_updated TIMESTAMP -Data update time
);

INSERT INTO products VALUES
(101, 'Phone', 2999.00, 50, NULL),
(102, 'PC', 5999.00, 30, NULL),
(103, 'PAD', 1999.00, 20, NULL);
```

Increase the price of all products by 20% (using a statement without **WHERE** to update the values of all records).

```
UPDATE products SET price = price * 1.2 , last_updated = NOW();
SELECT * FROM products ORDER BY 1;
```

| product_id | name  | price   | stock | last_updated               |
|------------|-------|---------|-------|----------------------------|
| 101        | Phone | 3598.80 | 50    | 2025-05-27 21:44:13.759399 |
| 102        | PC    | 7198.80 | 30    | 2025-05-27 21:44:13.759399 |
| 103        | PAD   | 2398.80 | 20    | 2025-05-27 21:44:13.759399 |

Increase the price of the PAD by 500 (using a statement with **WHERE** to update the specified data).

```
UPDATE products SET price = price + 500 , last_updated = NOW()
WHERE name = 'PAD';
SELECT * FROM products ORDER BY 1;
```

| product_id | name  | price   | stock | last_updated               |
|------------|-------|---------|-------|----------------------------|
| 101        | Phone | 3598.80 | 50    | 2025-05-27 21:44:13.759399 |
| 102        | PC    | 7198.80 | 30    | 2025-05-27 21:44:13.759399 |
| 103        | PAD   | 2898.80 | 20    | 2025-05-27 21:47:20.690807 |

Discount the price of products whose stock is greater than 40 by 20% (using a statement with **WHERE** to update data in a specified range).

```
UPDATE products SET price = price * 0.8 , last_updated = NOW()
WHERE stock > 40;
SELECT * FROM products ORDER BY 1;
```

| product_id | name  | price   | stock | last_updated               |
|------------|-------|---------|-------|----------------------------|
| 101        | Phone | 2879.04 | 50    | 2025-05-27 21:53:31.996656 |
| 102        | PC    | 7198.80 | 30    | 2025-05-27 21:44:13.759399 |
| 103        | PAD   | 2898.80 | 20    | 2025-05-27 21:47:20.690807 |

Adjust the price and stock of the PC (Specifying multiple values in **SET**)

```
UPDATE products SET price = price - 1000 , stock = stock - 10, last_updated = NOW()
WHERE name = 'PC';
SELECT * FROM products ORDER BY 1;
```

| product_id | name  | price   | stock | last_updated               |
|------------|-------|---------|-------|----------------------------|
| 101        | Phone | 2879.04 | 50    | 2025-05-27 21:53:31.996656 |
| 102        | PC    | 6198.80 | 20    | 2025-05-27 22:00:07.851013 |
| 103        | PAD   | 2898.80 | 20    | 2025-05-27 21:47:20.690807 |



## Example: Update a Specified Partition of a Partitioned Table

Create a sample table and import data into it.

```
DROP TABLE IF EXISTS test_range_row;
CREATE TABLE test_range_row(a int, b int,d int)
DISTRIBUTE BY hash(a) PARTITION BY RANGE(d)
(
 PARTITION p1 values LESS THAN (60),
 PARTITION p2 values LESS THAN (75),
 PARTITION p3 values LESS THAN (90),
 PARTITION p4 VALUES LESS THAN (maxvalue)
);

INSERT INTO test_range_row PARTITION(p1) VALUES(55,20,51);
INSERT INTO test_range_row PARTITION(p1) VALUES(54,18,59);
INSERT INTO test_range_row PARTITION(p2) VALUES(15,18,63);
INSERT INTO test_range_row PARTITION(p3) VALUES(12,21,80);
INSERT INTO test_range_row PARTITION(p3) VALUES(83,21,85);
INSERT INTO test_range_row PARTITION(p4) VALUES(36,21,97);

SELECT * FROM test_range_row ORDER BY d;
```

| a  | b  | d  |
|----|----|----|
| 55 | 20 | 51 |
| 54 | 18 | 59 |
| 15 | 18 | 63 |
| 12 | 21 | 80 |
| 83 | 21 | 85 |
| 36 | 21 | 97 |

Total Records: 6

Change the values of column **d** to **41** in partition **p1**.

```
UPDATE test_range_row PARTITION(p1) SET d = 41;
SELECT * FROM test_range_row ORDER BY d;
```

| a  | b  | d  |
|----|----|----|
| 54 | 18 | 41 |
| 55 | 20 | 41 |
| 15 | 18 | 63 |
| 12 | 21 | 80 |
| 83 | 21 | 85 |
| 36 | 21 | 97 |

Total Records: 6

According to the preceding result, the partition is **p1**, where the values of column **d** are less than **60**. The values of column **d** in partition **p1** are changed to **41**.

Update the values of column **b** to **20** in the partition where the partition key value is 41.

```
UPDATE test_range_row PARTITION FOR (41) SET b = 20;
SELECT * FROM test_range_row ORDER BY d;
```

| a  | b  | d  |
|----|----|----|
| 54 | 20 | 41 |
| 55 | 20 | 41 |
| 15 | 18 | 63 |
| 12 | 21 | 80 |
| 83 | 21 | 85 |
| 36 | 21 | 97 |

Total Records: 6

## Example: Use the FROM from\_list Clause to Preferentially Process Orders of Customers with High VIP Levels

The target table needs to be updated. The orders table needs to be updated, but the **FROM** clause needs to be used to associate the **customer ID** field in the **customers** table to obtain customer VIP levels.

Create the customer information table and order table, which contain the following fields:

```
DROP TABLE IF EXISTS customers;
CREATE TABLE customers (--Customer information table to be created
 customer_id INT PRIMARY KEY, -- Customer ID
 name VARCHAR(100), --Customer name
 vip_level INT --VIP level
);
DROP TABLE IF EXISTS orders;
CREATE TABLE orders (--Order table to be created
 order_id INT PRIMARY KEY, --Order ID
 customer_id INT, --Customer ID
 amount DECIMAL(10,2), --Consumption amount
 status VARCHAR(20) --Order status
);
```

Insert test data and ensure the order status of customers is pending (It means that the orders of the three customers are not processed.)

```
INSERT INTO customers VALUES
(1,'Tom', 1)
(2,'Jack', 3),
(3,'Tony', 2);
INSERT INTO orders VALUES
(1001, 1, 500.00, 'pending'),
(1002, 2, 1200.00, 'pending'),
(1003, 3, 800.00, 'pending');
SELECT * FROM orders;
```

| order_id | customer_id | amount  | status  |
|----------|-------------|---------|---------|
| 1002     | 2           | 1200.00 | pending |
| 1001     | 1           | 500.00  | pending |
| 1003     | 3           | 800.00  | pending |

Total Records: 3

Orders of customers with high VIP levels need be prioritized when there are too many orders. **FROM from\_list** is used to associate the customer ID in the customer table to obtain the customer VIP level. In this example, orders of customers with VIP level 3 or higher need to be processed first.

```
UPDATE orders o
SET status = 'priority' --Set the order status to priority.
FROM customers c
WHERE o.customer_id = c.customer_id AND c.vip_level >= 3; --Obtain the customers whose VIP levels are
 higher than 3 from the customer table.
SELECT * FROM orders;
```

| order_id | customer_id | amount  | status   |
|----------|-------------|---------|----------|
| 1002     | 2           | 1200.00 | priority |
| 1001     | 1           | 500.00  | pending  |
| 1003     | 3           | 800.00  | pending  |

Total Records: 3

## Example: Use DEFAULT to Reset the Status of an Inactive User

Create a user table (with DEFAULT) and insert data.

```
DROP TABLE IF EXISTS users;
CREATE TABLE users (
 user_id INT PRIMARY KEY, --User ID
 username VARCHAR(50), --Username
 login_count INT DEFAULT 0, -- Number of login times. The default value is 0.
 last_login TIMESTAMP, --Last login time
 status VARCHAR(20) DEFAULT 'active'; --User status. The default value is active.
)

INSERT INTO users (user_id, username, login_count, last_login) VALUES
(1, 'user1', 5, '2023-06-01 10:00:00'),
(2, 'user2', 3, '2023-06-15 14:30:00');
SELECT * FROM users;
```

| user_id | username | login_count | last_login          | status |
|---------|----------|-------------|---------------------|--------|
| 2       | user2    | 3           | 2023-06-15 14:30:00 | active |
| 1       | user1    | 5           | 2023-06-01 10:00:00 | active |

Total Records: 2

Reset the status and login times of inactive users:

```
UPDATE users SET
status = DEFAULT, -- Reset the status to active
login_count = DEFAULT -- Reset the login times to 0
WHERE last_login < CURRENT_DATE - INTERVAL '30 days';
SELECT * FROM users;
```

| user_id | username | login_count | last_login          | status |
|---------|----------|-------------|---------------------|--------|
| 1       | user1    | 0           | 2023-06-01 10:00:00 | active |
| 2       | user2    | 0           | 2023-06-15 14:30:00 | active |

## 21.14.11 VALUES

### Function

**VALUES** computes a row or a set of rows based on given values. It is most commonly used to generate a constant table within a large command.

### Precautions

- **VALUES** lists with large numbers of rows should be avoided, as you might encounter out-of-memory failures or poor performance. **VALUES** appearing within **INSERT** is a special case, because the desired column types are known from the **INSERT**'s target table, and need not be inferred by scanning the **VALUES** list. In this case, **VALUE** can handle larger lists than are practical in other contexts.
- If more than one row is specified, all the rows must have the same number of elements.

### Syntax

```
VALUES {(expression [, ...])} [, ...]
[ORDER BY { sort_expression [ASC | DESC | USING operator] } [, ...]]
[{ [LIMIT { count | ALL }] [OFFSET start [ROW | ROWS]] } | { LIMIT start, { count | ALL } }]
[FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY];
```

### Parameter Description

Table 21-293 VALUES parameters

| Parameter       | Description                                                                                                                                                                                                                                                                                                                                                                                                    | Value Range |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| expression      | Specifies a constant or expression to compute and insert at the indicated place in the resulting table or set of rows.<br><br>In a <b>VALUES</b> list appearing at the top level of an <b>INSERT</b> , an expression can be replaced by <b>DEFAULT</b> to indicate that the destination column's default value should be inserted. <b>DEFAULT</b> cannot be used when <b>VALUES</b> appears in other contexts. | -           |
| sort_expression | Specifies an expression or integer constant indicating how to sort the result rows.                                                                                                                                                                                                                                                                                                                            | -           |
| ASC             | Indicates ascending sort order.                                                                                                                                                                                                                                                                                                                                                                                | -           |
| DESC            | Indicates descending sort order.                                                                                                                                                                                                                                                                                                                                                                               | -           |
| operator        | Specifies a sorting operator.                                                                                                                                                                                                                                                                                                                                                                                  | -           |

| Parameter                                               | Description                                                                                                        | Value Range                     |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|---------------------------------|
| count                                                   | Specifies the maximum number of rows to return.                                                                    | -                               |
| start                                                   | Specifies the number of rows to skip before starting to return rows.                                               | -                               |
| FETCH { FIRST   NEXT } [ count ]<br>{ ROW   ROWS } ONLY | The <b>FETCH</b> clause restricts the total number of rows starting from the first row of the return query result. | The default value is <b>1</b> . |

## Inserting One or More Data Records

Create the **reason\_t1** table.

```
DROP TABLE IF EXISTS reason_t1;
CREATE TABLE reason_t1
(
 TABLE_SK INTEGER ,
 TABLE_ID VARCHAR(20),
 TABLE_NA VARCHAR(20)
);
```

Insert one data record to the table.

```
INSERT INTO reason_t1(TABLE_SK, TABLE_ID, TABLE_NA) VALUES (1, 'S01', 'StudentA');
```

Insert one data record into the table. This command is equivalent to the last one.

```
INSERT INTO reason_t1 VALUES (1, 'S01', 'StudentA');
```

Insert records whose **TABLE\_SK** is less than **1** into the table.

```
INSERT INTO reason_t1 SELECT * FROM reason_t1 WHERE TABLE_SK < 1;
```

Insert records into the table.

```
INSERT INTO reason_t1 VALUES (1, 'S01', 'StudentA'),(2, 'T01', 'TeacherA'),(3, 'T02', 'TeacherB');
SELECT * FROM reason_t1 ORDER BY 1;
```

| table_sk | table_id | table_na |
|----------|----------|----------|
| 1        | S01      | StudentA |
| 1        | S01      | StudentA |
| 1        | S01      | StudentA |
| 2        | T01      | TeacherA |
| 3        | T02      | TeacherB |

## 21.15 DCL Syntax

### 21.15.1 DCL Syntax Overview

Data control language (DCL) is used to set or modify database users or role rights.

#### Authorization

DWS provides a statement for granting rights to data objects and roles. For details, see [GRANT](#).

#### Revoking Rights

DWS provides a statement for revoking rights. For details, see [REVOKE](#).

#### Setting Default Rights

DWS allows users to set rights for objects that will be created. For details, see [ALTER DEFAULT PRIVILEGES](#).

### 21.15.2 ALTER DEFAULT PRIVILEGES

#### Function

**ALTER DEFAULT PRIVILEGES** allows you to set the permissions that will be used for objects to be created. It does not affect permissions assigned to existing objects.

A user can modify only the default permissions of objects created by the user or the role to which the user belongs. These permissions can be set globally (all objects created in the database) or for objects in a specified schema.

To view information about the default permissions of database users, query the system catalog **PG\_DEFAULT\_ACL**.

#### Precautions

- Only the permissions for tables (including views), sequences, functions, and types (including domains) can be altered.
- The VACUUM, DROP, and ALTER permissions on foreign tables cannot be granted to users.

#### Syntax

```
ALTER DEFAULT PRIVILEGES
[FOR { ROLE | USER } target_role [, ...]]
[IN SCHEMA schema_name [, ...]]
abbreviated_grant_or_revoke;
```

The **abbreviated\_grant\_or\_revoke** clause is used to specify the objects for which permissions are granted or revoked. The options include:

- `grant_on_tables_clause`
- `grant_on_functions_clause`
- `grant_on_types_clause`
- `grant_on_sequences_clause`
- `revoke_on_tables_clause`
- `revoke_on_functions_clause`
- `revoke_on_types_clause`
- `revoke_on_sequences_clause`

**grant\_on\_tables\_clause** grants permissions on tables.

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ANALYZE | ANALYSE
| VACUUM | ALTER | DROP }
[, ...] | ALL [PRIVILEGES] }
ON TABLES
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION]
```

**grant\_on\_functions\_clause** grants permissions on functions.

```
GRANT { { EXECUTE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
ON FUNCTIONS
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION]
```

**grant\_on\_types\_clause** grants permissions on types.

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON TYPES
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION]
```

**grant\_on\_sequences\_clause** grants permissions on sequences.

```
GRANT { { USAGE | SELECT | UPDATE }
[, ...] | ALL [PRIVILEGES] }
ON SEQUENCES
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION]
```

**revoke\_on\_tables\_clause** revokes permissions on tables.

```
REVOKE [GRANT OPTION FOR]
{ { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ANALYZE | ANALYSE |
VACUUM | ALTER | DROP }
[, ...] | ALL [PRIVILEGES] }
ON TABLES
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT | CASCADE CONSTRAINTS]
```

**revoke\_on\_functions\_clause** revokes permissions on functions.

```
REVOKE [GRANT OPTION FOR]
{ { EXECUTE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
ON FUNCTIONS
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT | CASCADE CONSTRAINTS]
```

**revoke\_on\_types\_clause** revokes permissions on types.

```
REVOKE [GRANT OPTION FOR]
{ USAGE | ALL [PRIVILEGES] }
ON TYPES
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT | CASCADE CONSTRAINTS]
```

**revoke\_on\_sequences\_clause** revokes permissions on sequences.

```
REVOKE [GRANT OPTION FOR]
{ { USAGE | SELECT | UPDATE }
[, ...] | ALL [PRIVILEGES] }
```



```
ON SEQUENCES
FROM { [GROUP] role_name | PUBLIC } [, ...]
[CASCADE | RESTRICT | CASCADE CONSTRAINTS]
```

## Parameter Description

Table 21-294 ALTER DEFAULT PRIVILEGES parameters

| Parameter                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FOR { ROLE   USER }<br>target_role<br>[, ...] | <p><b>target_role</b> following <b>FOR ROLE/USER</b> can be a role or user name that will create objects in the future. That is, the role or user name corresponding to the owner of the objects to be created in the future. If <b>FOR ROLE/USER</b> is omitted, the current role or user is assumed.</p> <p><b>target_role</b> must have the <b>CREATE</b> permissions for <b>schema_name</b>. You can use the <b>has_schema_privilege</b> function to check whether a role or user has the <b>CREATE</b> permission on a schema.</p> <pre>SELECT a.rolname, n.nspname FROM pg_authid<br/>as a, pg_namespace as n WHERE<br/>has_schema_privilege(a.oid, n.oid, 'CREATE');</pre> | <p>Name of an existing role or user.</p> <p>For example, grant the permission to user <b>jim</b> to access objects created by user <b>lily</b> in the future.</p> <pre>ALTER DEFAULT PRIVILEGES<br/>FOR ROLE lily IN SCHEMA<br/>lily GRANT SELECT ON<br/>tables TO jim;</pre> <p>or</p> <pre>ALTER DEFAULT PRIVILEGES<br/>FOR USER lily IN SCHEMA<br/>lily GRANT SELECT ON<br/>tables TO jim;</pre> |
| schema_name                                   | <p>Name of an existing schema.</p> <p>If a schema name is specified, the default permissions of all objects created in this schema will be modified. If <b>IN SCHEMA</b> is omitted, global permissions are modified.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                         | <p>Name of an existing schema.</p>                                                                                                                                                                                                                                                                                                                                                                  |
| role_name                                     | <p>Name of an existing role whose permissions are to be granted or revoked.</p> <p><b>NOTE</b></p> <p>If you want to delete a role that is granted with default permissions, you must revoke the default permissions or use <b>DROP OWNED BY</b> to delete the default permission records of the role.</p>                                                                                                                                                                                                                                                                                                                                                                        | <p>Name of an existing role.</p>                                                                                                                                                                                                                                                                                                                                                                    |

## Examples

- Grant the **SELECT** permission on all the tables (and views) in **tpcds** to every user.  

```
ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT SELECT ON TABLES TO PUBLIC;
```
- Grant the **INSERT** permission on all the tables in **tpcds** to user **jack**.  

```
ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds GRANT INSERT ON TABLES TO jack;
```

- Revoke the preceding permissions.  

```
ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE SELECT ON TABLES FROM PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA tpcds REVOKE INSERT ON TABLES FROM jack;
```
- Assume that there are two users **test1** and **test2**. If you require that user **test2** can query tables created by user **test1**, execute the following statements:
  - a. Grant the schema permission of user **test1** to **test2**.  

```
GRANT usage, create ON SCHEMA test1 TO test2;
```
  - b. Grant user **test2** the permission to query tables of user **test1**. **Note:** FOR USER can also be changed to FOR ROLE, but the effect is the same.  

```
ALTER DEFAULT PRIVILEGES FOR USER test1 IN SCHEMA test1 GRANT SELECT ON tables TO test2;
```
  - c. Create a table as user **test1**.  

```
SET ROLE test1 password '{password}';
CREATE TABLE test3(a int, b int);
```
  - d. Enable the **test2** user to perform the query.  

```
SET ROLE test2 password '{password}';
SELECT * FROM test1.test3;
a | b
---+---
(0 rows)
```

## Helpful Links

[GRANT, REVOKE](#)

## 21.15.3 ANALYZE | ANALYSE

### Function

This syntax gathers statistics on table content within a database. These statistics include both table-level and column-level details. Table-level data is kept in **PG\_CLASS**, while column-level details are stored in **PG\_STATISTIC**. The query planner relies on this information to choose the best execution strategy.

Its key purposes are:

- Optimizing query performance: supplies data distribution insights to the query optimizer for generating efficient plans.
- Enhancing plan accuracy: helps the optimizer better estimate the cost of various query paths.
- Selecting indexes: assists the optimizer in deciding if and which indexes to use.

### Scenarios Requiring the ANALYZE Operation

- **Significant data changes**  
Significant data changes, such as importing, updating, or deleting large amounts of data, alter row counts and data characteristics, requiring updated table statistics.
- **Newly added data querying**  
Daily updates to a service table with fresh data may lead to inaccurate queries if statistics are not refreshed promptly. The following tables have the characteristics of this scenario:

- Service tables that grow over time.
- Service tables that have daily data updates.
- Tables with fresh data that needs to be processed and analyzed.

Typically, after importing data, users process and analyze it using filters on partitioned tables.

```
passtime > '2020-01-19 00:00:00' AND pastime < '2020-01-20 00:00:00'
```

Without running **ANALYZE** after importing new data, the optimizer finds that the **passtime** filter range exceeds the recorded statistical limits. This leads to an incorrect row estimate, showing only one tuple matches the condition where **passtime** falls between 2020-01-19 00:00:00 and 2020-01-20 00:00:00.

## Precautions

- If no parameters are specified, **ANALYZE** analyzes each table and partitioned table in the database. You can also specify **table\_name**, **column**, and **partition\_name** to limit the analysis to a specified table, column, or partitioned table.
- Users who can execute **ANALYZE** on a specific table include the owner of the table, owner of the database where the table is, users with the **ANALYZE** permission on the table, users who are granted the **gs\_role\_analyze\_any** role, and users with the **SYSADMIN** attribute.
- To collect statistics using percentage sampling, you must have the **ANALYZE** and **SELECT** permissions.
- If the **enable\_analyze\_partition** parameter is enabled and the table-level **incremental\_analyze** parameter is set for a partitioned table, the **ANALYZE** statement is executed on partitions lacking statistics or with data changes. The statistics of the partition main table are then generated by combining the partition statistics.
- Only cluster versions 8.1.1 and later support using anonymous blocks, transaction blocks, functions, or stored procedures to perform the **ANALYZE** operation on an unsharded table.
- However, for analyzing an entire database, the **ANALYZE** operation of each table is in different transactions. Therefore, the current version does not support the **ANALYZE** execution for the entire database in anonymous blocks, transaction blocks, functions, or stored procedures.
- Statistics updates of PG\_CLASS related columns cannot be rolled back.
- The **ANALYZE** statistics of global temporary tables in each session are independent. Each session has its own statistics.
- **ANALYZE** and **ANALYZE VERIFY** are used to check whether data files of common tables (row-store and column-store tables) in a database are damaged. Currently, this function does not support HDFS tables.
- Most **ANALYZE VERIFY** operations are used for abnormal scenario detection, and require a release version. Remote read is not triggered in the **ANALYZE VERIFY** scenario. Therefore, the remote read parameter does not take effect. If the system detects that the page is damaged due to an error in the key system table, the system reports an error and stops the detection.

**WARNING**

- If more than 10% of a table's records are added or modified at once, explicitly execute the **ANALYZE** operation.
- For more information about development and design specifications, see "DWS Development and Design Proposal" in the *Developer Guide*.

## Syntax

- Collect statistics information about a table.  

```
{ ANALYZE | ANALYZE } [{ VERBOSE | LIGHT | FORCE | PREDICATE }]
[table_name [(column_name [, ...])]] ;
```
- Collect statistics about a partition.  

```
{ ANALYZE | ANALYZE } [{ VERBOSE | LIGHT | FORCE }]
[table_name [(column_name [, ...])]]
PARTITION (partition_name) ;
```

  - Ordinary partitioned tables support the syntax for collecting statistics on a specific partition, but not the actual function of it. **ANALYZE** on a specified partition will cause a warning message.
  - Temporary sampling tables cannot be used to collect partition statistics.
  - Multi-column statistics and expression statistics on partitions are not supported.
- Collect statistics about a foreign table.  

```
{ ANALYZE | ANALYZE } [VERBOSE]
{ foreign_table_name | FOREIGN TABLES } ;
```
- Collect statistics about multiple columns.  

```
{ANALYZE | ANALYZE} [VERBOSE]
table_name ((column_1_name, column_2_name [, ...]));
```

  - To sample data in percentage, set **default\_statistics\_target** to a negative number.
  - The statistics about a maximum of 32 columns can be collected at a time.
  - You are not allowed to collect statistics about multiple columns in system catalogs, global temporary tables, or HDFS foreign tables.
- Check the data files in the current database.  

```
{ANALYZE | ANALYZE} VERIFY {FAST|COMPLETE};
```

  - All operations on the database are supported. Because many tables are involved, you are advised to save the result in redirection mode: **gsql -d database -p port -f "verify.sql"> verify\_warning.txt 2>&1**.
  - HDFS tables (internal and foreign tables), temporary tables, and unlog tables are not supported.
  - Only the NOTICE message is displayed for external tables. The detection of internal tables is included in the external tables that depend on them, but it is not shown externally.
  - This command can be used to process tolerant errors. The assert operation in a debug version may cause the core to fail to execute commands. Therefore, you are advised to perform this operation in a release version.
  - If a key system table is damaged during a full database operation, an error is reported and the operation stops.

- Check data files of tables and indexes.  
`{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name[index_name] [CASCADE];`
  - You can perform operations on common tables and index tables, but cannot perform **CASCADE** operations on index tables. The reason is that **CASCADE** is used to process all index tables of the primary table. When the index table is checked separately, **CASCADE** is not required.
  - HDFS tables (internal and foreign tables), temporary tables, and unlog tables are not supported.
  - When the primary table is checked, the internal tables of the primary table, such as the **toast** table and **cudesc** table, are also checked.
  - When the system displays a message indicating that the index table is damaged, you are advised to run the **reindex** command to recreate the index.
- Check the data files of the table partition.  
`{ANALYZE | ANALYSE} VERIFY {FAST|COMPLETE} table_name PARTITION {(partition_name)} [CASCADE];`
  - You can detect a single partition of a table, but cannot perform the **CASCADE** operation on index tables.
  - HDFS tables (internal and foreign tables), temporary tables, and unlog tables are not supported.

## Parameter Description

Table 21-295 ANALYZE | ANALYSE parameters

| Parameter | Description                                                                                                                                                                                                                                           | Value Range |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| VERBOSE   | After <b>VERBOSE</b> is enabled, <b>ANALYZE</b> sends progress information, indicating the table being processed. Statistics about the table are also printed.                                                                                        | -           |
| LIGHT     | In <b>LIGHT</b> mode, the system stores table statistics in memory instead of writing them to the system catalog. Executing this command locks the table at a certain level. Specify this option within parentheses, like <b>ANALYZE (LIGHT) t1</b> . | -           |
| FORCE     | In <b>FORCE</b> mode, you can override locks on table statistics and run <b>ANALYZE</b> . Add this option in parentheses like this: <b>ANALYZE (FORCE) t1</b> .                                                                                       | -           |

| Parameter          | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range                                                                                                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PREDICATE          | <p><b>PREDICATE</b> indicates that only the statistics of the currently identified predicate column are calculated. This parameter is supported only by clusters of version 9.1.0.100 or later. You can set the <b>analyze_predicate_column_threshold</b> parameter to determine whether the <b>ANALYZE</b> operation on predicate columns is supported. A value of <b>0</b> indicates that the <b>ANALYZE</b> operation is disabled. A value greater than <b>0</b> indicates that the <b>ANALYZE</b> operation is enabled only for tables whose number of columns is greater than or equal to this value.</p> <p>Use this option in parentheses, for example, <b>ANALYZE (PREDICATE) xxx</b>.</p> <p>Predicate information is collected in the query parsing phase. Dynamic sampling also supports predicate column sampling.</p> | -                                                                                                                                                                                          |
| table_name         | <p>Specifies the name (possibly schema-qualified) of a specific table to analyze. If omitted, all regular tables (but not foreign tables) in the current database are analyzed.</p> <p>Currently, you can use <b>ANALYZE</b> to collect statistics from row-store tables, column-store tables, HDFS tables, ORC- and CARBONDATA-formatted OBS foreign tables, and foreign tables for collaborative analysis.</p>                                                                                                                                                                                                                                                                                                                                                                                                                   | The table name can contain a maximum of 63 characters, including letters, digits, underscores (_), dollar signs (\$), and number signs (#). It must start with a letter or underscore (_). |
| column_name        | Specifies the name of a specific column to analyze. All columns are analyzed by default.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Existing column name (format: <b>column_1_name, column_2_name, column_3_name....</b> )                                                                                                     |
| partition_name     | Analyzes the statistics of the partitioned table. Currently the partitioned table supports the syntax of analyzing a partitioned table, but does not execute this syntax.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | A string, which must comply with the naming convention of the partition name.                                                                                                              |
| foreign_table_name | Specifies the name (possibly schema-qualified) of a specific table to analyze. The data of the table is stored in HDFS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | A string, which must comply with the naming convention of foreign tables.                                                                                                                  |

| Parameter      | Description                                                                     | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FOREIGN TABLES | Analyzes HDFS foreign tables stored in HDFS and accessible to the current user. | A string, which must comply with the naming rules of HDFS foreign tables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| index_name     | Name of the index table to be analyzed, which may contain the schema name.      | A string, which must comply with the naming rules of index tables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| FAST COMPLETE  | Verifies row-store tables and column-store tables in different modes.           | <ul style="list-style-type: none"><li>• Row-store tables<ul style="list-style-type: none"><li>– In FAST mode, the CRC and page header of the row-store table are verified. If the verification fails, an alarm is generated.</li><li>– In COMPLETE mode, the pointer and tuple of the row-store table are parsed and verified.</li></ul></li><li>• Column-store tables<ul style="list-style-type: none"><li>– In FAST mode, the CRC and magic of the column-store table are verified. If the verification fails, an alarm is generated.</li><li>– In COMPLETE mode, CUs of column-store tables are parsed and verified.</li></ul></li></ul> |
| CASCADE        | In <b>CASCADE</b> mode, all indexes of the current table are checked.           | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## Examples

- Do **ANALYZE** to update statistics in the **customer\_info** table:  
`ANALYZE customer_info;`
- Do **ANALYZE VERBOSE** to update statistics and display table information in the **customer\_info** table:  
`ANALYZE VERBOSE customer_info;`  
INFO: analyzing "cstore.pg\_delta\_3394584009"(cn\_5002 pid=53078)  
INFO: analyzing "public.customer\_info"(cn\_5002 pid=53078)  
INFO: analyzing "public.customer\_info" inheritance tree(cn\_5002 pid=53078)  
`ANALYZE`
- Update the statistics of the **name** and **email** columns in the **customer\_info** table.  
`ANALYZE customer_info (name, email);`
- Update the statistics of the **customer\_info** table in lightweight mode.  
`ANALYZE (LIGHT) customer_info;`
- Update the statistics of the **customer\_info** table in forcible mode, even if the table is locked.  
`ANALYZE (FORCE) customer_info;`
- Update only the statistics of predicate columns. Only the predicate columns collected so far are sampled. The statistics are used after the query service is stable.  
`ANALYZE (PREDICATE) customer_info;`  

You can use the `pg_stat_get_predicate_columns` function to query the predicate columns of a table. The table name in the following example is **customer\_info**. Replace it with the actual table name.

```
SELECT pr.attnum,pa.attname FROM
pg_catalog.pg_stat_get_predicate_columns('customer_info'::regclass) pr LEFT JOIN pg_attribute pa ON
pa.attrelid='customer_info'::regclass AND pa.attnum = pr.attnum;
```

## 21.15.4 DEALLOCATE

### Function

Deallocates a previously prepared statement. If a prepared statement is not explicitly deleted, it is deleted at the end of the session.

For details about prepared statements, see [PREPARE](#).

### Precautions

None

### Syntax

```
DEALLOCATE [PREPARE] { name | ALL };
```



## Parameter Description

Table 21-296 DEALLOCATE parameters

| Parameter | Description                                                                            | Value Range                                                               |
|-----------|----------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| PREPARE   | Creates a prepared statement. This keyword is optional and does not affect operations. | -                                                                         |
| name      | Specifies the name of the prepared statement to be deleted.                            | A string, which must comply with the naming rules of prepared statements. |
| ALL       | Deallocates all prepared statements.                                                   | A string, which must comply with the prepared statement format.           |

## Examples

None

## 21.15.5 DO

### Function

**DO** executes an anonymous code block.

A code block is a function body without parameters. Its return type is **void**. It is analyzed and executed at the same time.

### Precautions

- Before using a programming language, install it in the current database using **CREATE LANGUAGE**. If no language is specified, **plpgsql** is installed by default.
- To use an untrusted language, you must be a system administrator or have the USAGE permission for programming languages.

### Syntax

```
DO [LANGUAGE lang_name] code;
```

## Parameter Description

Table 21-297 DO parameters

| Parameter | Description                                          | Value Range                                                                               |
|-----------|------------------------------------------------------|-------------------------------------------------------------------------------------------|
| lang_name | Name of the programming language used to parse code. | Program language that complies with the specifications. The default language is PL/pgSQL. |
| code      | Programming language code that can be executed.      | A string.                                                                                 |

## Examples

Grant user **webuser** all the operation permissions on views in the **tpcds** schema:

```
DO $$DECLARE r record;
BEGIN
 FOR r IN SELECT c.relname,n.nspname FROM pg_class c,pg_namespace n
 WHERE c.relnamespace = n.oid AND n.nspname = 'tpcds' AND relkind IN ('r','v')
 LOOP
 EXECUTE 'GRANT ALL ON ' || quote_ident(r.table_schema) || '.' || quote_ident(r.table_name) || ' TO
webuser';
 END LOOP;
END$$;
```

## 21.15.6 EXECUTE

### Function

**EXECUTE** executes a prepared statement. Because a prepared statement exists only in the lifetime of a session, the prepared statement must be created by an earlier **PREPARE** statement in the current session.

### Precautions

If the **PREPARE** statement creating the prepared statement declares certain parameters, the parameter set transferred to the **EXECUTE** statement must be compatible. Otherwise, an error occurs.

### Syntax

```
EXECUTE name [(parameter [, ...])] ;
```

## Parameter Description

Table 21-298 EXECUTE parameters

| Parameter | Description                                                                                                                                                                      | Value Range                                                                                                   |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|
| name      | Specifies the name of the statement to be executed.                                                                                                                              | A string, which must comply with the naming rules of prepared statements.                                     |
| parameter | Specifies a parameter of the prepared statement. It must be an expression that generates a value compatible with the data type specified when the prepared statement is created. | A string, which is an expression of a value that is compatible with the data type of the specified parameter. |

## Examples

Create and run a prepared statement for the **INSERT** statement:

```
PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO tpcds.reason_t1
VALUES($1,$2,$3);
EXECUTE insert_reason(52, 'AAAAAAADDDAAAAA', 'reason 52');
```

## 21.15.7 EXECUTE DIRECT

### Function

**EXECUTE DIRECT** executes an SQL statement on a specified node. Generally, the cluster automatically allocates an SQL statement to proper nodes. **EXECUTE DIRECT** is mainly used for database maintenance and testing.

### Precautions

- Only a system administrator can run the **EXECUTE DIRECT** statement.
- To ensure data consistency across nodes, only the **SELECT** statement can be used. Transaction statements, DDL, and DML cannot be used.
- When the AVG aggregation calculation is performed on the specified DN using such statements, the result set is returned in array, for example, {4,2}. The result of sum is 4, and that of count is 2.
- Do not run the **SELECT** statement on nodes where CNs reside because user table data is not stored there.
- **EXECUTE DIRECT** cannot be nested. If the inner SQL statement to be executed is also **EXECUTE DIRECT**, run only the bottom-layer **EXECUTE DIRECT** statement.

## Syntax

```
EXECUTE DIRECT ON (nodename [, ...]) query ;
```

## Parameter Description

**Table 21-299** EXECUTE DIRECT parameters

| Parameter | Description              | Value Range                                                   |
|-----------|--------------------------|---------------------------------------------------------------|
| nodename  | Node name                | A string that indicates the name of an existing node.         |
| query     | Statement to be executed | A string that indicates a SQL statement that can be executed. |

## Examples

Query records in table **tpcds.customer\_address** on the dn\_6001\_6002 node:

```
EXECUTE DIRECT ON(dn_6001_6002) 'select count(*) from tpcds.customer_address';
count

16922
(1 row)
```

## 21.15.8 GRANT

### Functions and Precautions

**GRANT** grants permissions to roles and users.

**GRANT** is used in the following scenarios:

### Granting System Permissions (SYSADMIN, CREATEDB, CREATEROLE, AUDITADMIN, and LOGIN) to Roles or Users

System permissions are also called user attributes, including **SYSADMIN**, **CREATEDB**, **CREATEROLE**, **AUDITADMIN**, and **LOGIN**.

**Table 21-300** System permissions

| System Permission | Description                                 | Can Be Granted or Revoked |
|-------------------|---------------------------------------------|---------------------------|
| <b>SYSADMIN</b>   | Allows you to view special system catalogs. | Yes                       |
| <b>CREATEDB</b>   | Allows you to creates databases.            | No                        |
| <b>CREATEROLE</b> | Allow you to create users and roles.        |                           |

| System Permission | Description                         | Can Be Granted or Revoked |
|-------------------|-------------------------------------|---------------------------|
| <b>AUDITADMIN</b> | Allows you to view audit logs.      |                           |
| <b>LOGIN</b>      | Allows you to connect to databases. |                           |

System permissions are generally specified using **CREATE ROLE** or **ALTER ROLE** syntax.

The **SYSADMIN** permission can be granted and revoked using **GRANT ALL PRIVILEGE** and **REVOKE ALL PRIVILEGE**, respectively.

System permissions cannot be inherited by a user from a role, and cannot be granted using **PUBLIC**.

## Granting Database Objects (Tables, Views, Specified Columns, Databases, Functions, and Schemas) to Roles or Users

### Precautions

- Do not perform **GRANT** on a table partition. Otherwise, an alarm will be generated.
- To rename an existing object, you must own the object and have the **CREATE** permission on the schema of the object.
- When you grant table or view rights to other users, you also need to grant the **USAGE** permission for the schema that the tables and views belong to. Without this permission, the users granted with the table or view rights can only see the object names, but cannot access them.
- **GRANT** grants specified database object permissions to one or more roles. These permissions are appended to those already granted, if any.
- The key word **PUBLIC** indicates that the permissions are to be granted to all roles, including those that might be created later. **PUBLIC** can be regarded as an implicitly defined group including all roles. Any particular role will have the sum of permissions granted directly to it using **GRANT**, permissions granted to any role it is presently a member of, and permissions granted to **PUBLIC**.
- If **WITH GRANT OPTION** is specified, the recipient of a permission can in turn grant it to others. This option cannot be granted to **PUBLIC**. Only DWS supports this operation.
- DWS grants the permissions for objects of certain types to **PUBLIC**. By default, permissions for tables, table columns, sequences, external data sources, external servers, schemas, and tablespace are not granted to **PUBLIC**. However, permissions for the following objects are granted to **PUBLIC**: **CONNECT** and **CREATE TEMP TABLE** permissions for databases, **EXECUTE** permission for functions, and **USAGE** permission for languages and data types (including domains). An object owner can revoke the default permissions granted to **PUBLIC** and grant permissions to other users as needed. For security purposes, you are advised to create an object and set permissions for

it in the same transaction so that other users do not have time windows to use the object. In addition, you can run the **ALTER DEFAULT PRIVILEGES** statement to modify the initial default permissions.

## Granting the Permissions of a Role or r User to Others

Grant a role's or user's permissions to one or more roles or users. In this case, every role or user can be regarded as a set of one or more database permissions.

- If **WITH ADMIN OPTION** is specified, the member can in turn grant permissions in the role to others, and revoke permissions in the role as well.
- If a role or user granted with certain permissions is changed or revoked, the permissions inherited from the role or user also change.
- A database administrator can grant permissions to and revoke them from any role or user.
- Roles having **CREATEROLE** permission can grant or revoke membership in any role that is not an administrator.

## Permissions

**Table 21-301** GRANT permissions

| Parameter         | Description                                                                                                                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SELECT            | Allows <b>SELECT</b> from any column, or the specific columns listed, of the specified table, view, or sequence.                                                                                                                                    |
| INSERT            | Allows <b>INSERT</b> of a new row into the specified table.                                                                                                                                                                                         |
| UPDATE            | Allows <b>UPDATE</b> of any column, or the specific columns listed, of the specified table. <b>SELECT ... FOR UPDATE</b> and <b>SELECT ... FOR SHARE</b> also require this permission on at least one column, in addition to the SELECT permission. |
| DELETE            | Allows <b>DELETE</b> of a row from the specified table.                                                                                                                                                                                             |
| TRUNCATE          | Allows <b>TRUNCATE</b> on the specified table.                                                                                                                                                                                                      |
| REFERENCES        | To create a foreign key constraint, it is necessary to have this permission on both the referencing and referenced columns.                                                                                                                         |
| TRIGGER           | To create a trigger, you must have the TRIGGER permission on the table or view.                                                                                                                                                                     |
| ANALYZE   ANALYSE | To perform the ANALYZE   ANALYSE operation on a table to collect statistics data, you must have the ANALYZE   ANALYSE permission on the table.                                                                                                      |

| Parameter        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CREATE           | <ul style="list-style-type: none"><li>For databases, allows new schemas to be created within the database.</li><li>For schemas, allows new objects to be created within the schema. To rename an existing object, you must own the object and have the <b>CREATE</b> permission on the schema of the object.</li><li>For sub-clusters, allows tables to be created.</li></ul>                                                                                                                                                      |
| CONNECT          | Allows the user to connect to the specified database.                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| TEMPORARY   TEMP | Allows temporary tables to be created when the specified database is used.                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| EXECUTE          | Allows the use of the specified function and the use of any operators that are implemented on top of the function.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| USAGE            | <ul style="list-style-type: none"><li>Procedural languages: It allows the use of the specified language for the creation of functions in that language.</li><li>Schemas: It allows access to objects contained in the specified schema. Without this permission, it is still possible to see the object names.</li><li>Sequences: It allows the use of the nextval function.</li><li>Sub-clusters: It allows users who can access objects contained in the specified schema to access tables in a specified sub-cluster.</li></ul> |
| COMPUTE          | Allows users to perform elastic computing in a computing sub-cluster that they have the compute permission on.                                                                                                                                                                                                                                                                                                                                                                                                                     |
| ALTER            | Allows you to modify tables, schemas, or functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| DROP             | Allows you to delete tables, schemas, or functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| VACUUM           | Allows <b>VACUUM</b> to be executed on tables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| ALL PRIVILEGES   | Grants all of the available permissions at once. Only system administrators have permission to run <b>GRANT ALL PRIVILEGES</b> .                                                                                                                                                                                                                                                                                                                                                                                                   |

| Parameter         | Description                                                                                                                                                                                                                                                                                                                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WITH GRANT OPTION | If <b>WITH GRANT OPTION</b> is specified, the recipient of a permission can in turn grant it to others. This option cannot be granted to <b>PUBLIC</b> . <ul style="list-style-type: none"><li>• <b>WITH GRANT OPTION</b> cannot be used with <b>NODE GROUP</b>.</li><li>• When using <b>WITH GRANT OPTION</b>, ensure that <b>grant_with_grant_option</b> is set for <b>security_enable_options</b>.</li></ul> |
| WITH ADMIN OPTION | Specifies whether permission transfer is allowed. If <b>WITH ADMIN OPTION</b> is specified, members of a role can grant membership of the role to others.                                                                                                                                                                                                                                                       |

## Syntax

- Grant the table or view access permission to a specified role or user. Do not perform **GRANT** on a table partition. Otherwise, an alarm will be generated.  
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ANALYZE | ANALYSE | VACUUM | ALTER | DROP } [, ...]  
| ALL [ PRIVILEGES ] }  
ON { [ TABLE ] table\_name [, ...]  
| ALL TABLES IN SCHEMA schema\_name [, ...] }  
TO { [ GROUP ] role\_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];

### CAUTION

Do not perform concurrent GRANT or REVOKE operations (including **ON TABLE** and **ON ALL TABLES IN SCHEMA**) on a given table or view, or do not perform both GRANT or REVOKE and DDL operations on a given table or view. Otherwise, an error similar to "tuple concurrently updated." is reported.

- Grant the column access permission to a specified role or user.  
GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column\_name [, ...] ) } [, ...]  
| ALL [ PRIVILEGES ] ( column\_name [, ...] ) }  
ON [ TABLE ] table\_name [, ...]  
TO { [ GROUP ] role\_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];
- Grant the schema access permission to a specified role or user.  
GRANT { { CREATE | USAGE | ALTER | DROP } [, ...] | ALL [ PRIVILEGES ] }  
ON SCHEMA schema\_name [, ...]  
TO { [ GROUP ] role\_name | PUBLIC } [, ...]  
[ WITH GRANT OPTION ];

### CAUTION

When you grant table or view rights to other users, you also need to grant the USAGE permission for the schema that the tables and views belong to. Without this permission, the users granted with the table or view rights can only see the object names, but cannot access them.



- Grant the database access permission to a specified role or user.

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...]
 | ALL [PRIVILEGES] }
ON DATABASE database_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant a role's permissions to other users or roles.

```
GRANT role_name [, ...]
TO role_name [, ...]
[WITH ADMIN OPTION];
```

- Grant all permissions to a specified role.

```
GRANT ALL { PRIVILEGES | PRIVILEGE }
TO role_name;
```

- Grant the domain access permission to a specified role or user.

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON DOMAIN domain_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

#### NOTE

The current version does not support granting the domain access permission.

- Grant the external data source access permission to a specified role or user.

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON FOREIGN DATA WRAPPER fdw_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the external server access permission to a specified role or user.

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON FOREIGN SERVER server_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the function access permission to a specified role or user.

```
GRANT { { EXECUTE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
ON { FUNCTION {function_name ([{[argmode] [arg_name] arg_type } [, ...]) } [, ...]
 | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the procedural language access permission to a specified role or user.

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON LANGUAGE lang_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

#### NOTE

The current version does not support granting the procedural language access permission.

- Grant the large object access permission to a specified role or user.

```
GRANT { { SELECT | UPDATE } [, ...] | ALL [PRIVILEGES] }
ON LARGE OBJECT loid [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

#### NOTE

The current version does not support granting the large object access permission.

- Grant the sequence access permission to a specified role or user.

```
GRANT { { SELECT | UPDATE | USAGE } [, ...] | ALL [PRIVILEGES] }
ON { SEQUENCE sequence_name [, ...]
 | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the sub-cluster access permission to a specified role or user. Common users cannot perform **GRANT** or **REVOKE** operations on node groups.

```
GRANT { CREATE | USAGE | COMPUTE | ALL [PRIVILEGES] }
ON NODE GROUP group_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

- Grant the type access permission to a specified role or user.

```
GRANT { USAGE | ALL [PRIVILEGES] }
ON TYPE type_name [, ...]
TO { [GROUP] role_name | PUBLIC } [, ...]
[WITH GRANT OPTION];
```

 **NOTE**

The current version does not support granting the type access permission.

## Parameter Description

Table 21-302 GRANT parameters

| Parameter     | Description                                                                                                                 | Value Range                                                                                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| role_name     | Specifies an existing user name. To obtain the value, see PG_USER.                                                          | A string, which must comply with the naming convention. Enter a maximum of 63 characters.                                                                                                                     |
| table_name    | Specifies an existing table name. To obtain the value, see PG_TABLES.                                                       | A string of no more than 63 characters, starting with a letter or underscore (_). The value can contain letters, digits, underscores (_), dollar signs (\$), and number signs (#).                            |
| column_name   | Specifies an existing column name. To obtain the value, run:<br><pre>SELECT * FROM<br/>PG_GET_TABLEDEF('table name');</pre> | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                      |
| schema_name   | Specifies an existing schema name. To obtain the value, see PG_NAMESPACE.                                                   | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                      |
| database_name | Specifies an existing database name. To obtain the value, PG_DATABASE.                                                      | A string that complies with the naming convention, which includes uppercase and lowercase letters, digits, underscores (_), and dollar signs (\$). However, it cannot begin with a digit or dollar sign (\$). |

| Parameter     | Description                                                                                                                      | Value Range                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| function_name | Specifies an existing function name. To obtain the value, PG_PROC.                                                               | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| sequence_name | Specifies a sequence schema name. To obtain the value, view PG_SEQUENCES (supported only by clusters of version 9.1.0 or later). | A string that contains only letters (both uppercase and lowercase), special characters #, \$, and digits.                |
| domain_name   | Specifies an existing domain type.                                                                                               | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| fdw_name      | Specifies an existing foreign data wrapper name.                                                                                 | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| lang_name     | Specifies an existing language name.                                                                                             | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| type_name     | Specifies an existing type name.                                                                                                 | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| group_name    | Specifies an existing sub-cluster name.                                                                                          | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| argmode       | Specifies the parameter mode of a function.                                                                                      | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| arg_name      | Specifies the parameter name of a function.                                                                                      | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |
| arg_type      | Specifies the parameter type of a function.                                                                                      | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |

| Parameter | Description                                                           | Value Range                                                                                                              |
|-----------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| loid      | Specifies the identifier of the large object that contains this page. | A string, which must comply with the naming convention. For details, see <a href="#">Identifier Naming Conventions</a> . |

## Examples

- **Grant system permissions to a user or role.**
  - Grant all available permissions of user **sysadmin** to user **joe**:  

```
GRANT ALL PRIVILEGES TO joe;
```

Afterward, user **joe** has the sysadmin permissions.
- **Grant object permissions to a user or role.**
  - Grant the SELECT permission on the **tpcds.reason** table to user **joe**:  

```
GRANT SELECT ON TABLE tpcds.reason TO joe;
```
  - Grant all permissions of the **tpcds.reason** table to user **kim**.  

```
GRANT ALL PRIVILEGES ON tpcds.reason TO kim;
```

After the granting succeeds, user **kim** has all the permissions of the **tpcds.reason** table, including the add, delete, modify, and query permissions.
  - Grant the permission to use the **tpcds** schema to user **joe**.  

```
GRANT USAGE ON SCHEMA tpcds TO joe;
```

After the authorization is successful, user **joe** has the **USAGE** permission of the schema and can access the objects contained in the schema.
  - Grant the query permission for the **r\_reason\_sk**, **r\_reason\_id**, and **r\_reason\_desc** columns and the update permission for the **r\_reason\_desc** column in the **tpcds.reason** table to user **joe**:  

```
GRANT select (r_reason_sk,r_reason_id,r_reason_desc),update (r_reason_desc) ON tpcds.reason TO joe;
```

After the granting succeeds, user **joe** immediately has the query permission of the **r\_reason\_sk** and **r\_reason\_id** columns in the **tpcds.reason** table.

```
GRANT select (r_reason_sk, r_reason_id) ON tpcds.reason TO joe ;
```
  - Grant the **EXECUTE** permission of the **func\_add\_sql** function to user **joe**.  

```
CREATE FUNCTION func_add_sql(f1 integer,f2 integer) RETURNS integer
AS 'select $1 + $2;'
LANGUAGE SQL
IMMUTABLE
RETURNS NULL ON NULL INPUT;
GRANT EXECUTE ON FUNCTION func_add_sql(integer, integer) TO joe;
```
  - Grant the **UPDATE** permission of the sequence **serial** to user **joe**.  

```
GRANT UPDATE ON SEQUENCE serial TO joe;
```
  - Grant user **joe** the database connection permission of gaussdb and the permission to create schemas in gaussdb.  

```
GRANT create,connect on database gaussdb TO joe ;
```
  - Grant the **tpcds** schema access permission and object creation permission to this role, but do not enable it to grant these permissions to others:  

```
GRANT USAGE,CREATE ON SCHEMA tpcds TO tpcds_manager;
```
- **Grant the permissions of a user or role to other users or roles.**

- Grant the permissions of user **joe** to user **manager**, and allow **manager** to grant these permissions to others:  
`GRANT joe TO manager WITH ADMIN OPTION;`
- Grant the permissions of user **manager** to user **senior\_manager**:  
`GRANT manager TO senior_manager;`

## Helpful Links

[REVOKE](#), [ALTER DEFAULT PRIVILEGES](#)

## 21.15.9 PREPARE

### Function

**PREPARE** creates a prepared statement.

A prepared statement is a performance optimizing object on the server. When the **PREPARE** statement is executed, the specified query is parsed, analyzed, and rewritten. When the **EXECUTE** is executed, the prepared statement is planned and executed. This avoids repetitive parsing and analysis. After the **PREPARE** statement is created, it exists throughout the database session. Once it is created (even if in a transaction block), it will not be deleted when a transaction is rolled back. It can only be deleted by explicitly invoking [DEALLOCATE](#) or automatically deleted when the session ends.

### Precautions

None

### Syntax

```
PREPARE name [(data_type [, ...])] AS statement;
```

### Parameter Description

Table 21-303 PREPARE parameters

| Parameter | Description                                                     | Value Range                                                            |
|-----------|-----------------------------------------------------------------|------------------------------------------------------------------------|
| name      | Name of a prepared statement. It must be unique in the session. | A string                                                               |
| data_type | Data type of the parameter.                                     | -                                                                      |
| statement | SQL statement that can be executed.                             | Any <b>SELECT INSERT, UPDATE, DELETE</b> , or <b>VALUES</b> statement. |

### Examples

Create and run a prepared statement for the **INSERT** statement:

```
PREPARE insert_reason(integer,character(16),character(100)) AS INSERT INTO tpcds.reason_t1
VALUES($1,$2,$3);
EXECUTE insert_reason(52, 'AAAAAAAADDAAAAAA', 'reason 52');
```

## Helpful Links

[DEALLOCATE](#)

## 21.15.10 REASSIGN OWNED

### Function

**REASSIGN OWNED** changes the owner of a database.

**REASSIGN OWNED** requires that the system change owners of all the database objects owned by **old\_role** to **new\_role**.

### Precautions

- **REASSIGN OWNED** is often executed before deleting a rule. Because objects in other databases are not affected, you usually need to run this command in each database that contains the objects owned by the role to be deleted.
- You must have the permissions on the original and target roles to execute it.
- The resource management module does not monitor the data switch of the syntax. You need to call **select gs\_wlm\_readjust\_user\_space(0)** to manually calibrate the monitoring data.

### Syntax

```
REASSIGN OWNED BY old_role [, ...] TO new_role;
```

### Parameter Description

**Table 21-304** REASSIGN OWNED parameters

| Parameter | Description                                             | Value Range                                                                                                                                                             |
|-----------|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| old_role  | Role name of the old owner.                             | A string of no more than 63 characters. Enter a string starting with a letter or underscore (_) and containing letters, digits, underscores (_), and dollar signs (\$). |
| new_role  | Name of the new role that will become the object owner. | A string of no more than 63 characters. Enter a string starting with a letter or underscore (_) and containing letters, digits, underscores (_), and dollar signs (\$). |

### Examples

Reassign all database objects owned by the **joe** and **jack** roles to **admin**:

```
REASSIGN OWNED BY joe, jack TO admin;
```

## 21.15.11 REVOKE

### Function

**REVOKE** revokes rights from one or more roles.

### Precautions

If a non-owner user of an object attempts to **REVOKE** rights on the object, the command is executed based on the following rules:

- If the user has no right whatsoever on the object, the command will fail outright.
- If some permissions are available, the command proceeds, but it revokes only those rights for which the user has grant options.
- The **REVOKE ALL PRIVILEGES** forms will issue an error message if no grant options are held, while the other forms will issue a warning if grant options for any of the rights named in the command are not held.
- Do not perform **REVOKE** to a table partition. Performing **REVOKE** to a partitioned table incurs an alarm.

### Syntax

- Revoke the permission of specified table and view.

```
REVOKE [GRANT OPTION FOR]
 { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ANALYZE |
 ANALYZE | VACUUM | ALTER | DROP }[, ...]
 | ALL [PRIVILEGES] }
 ON { [TABLE] table_name [, ...]
 | ALL TABLES IN SCHEMA schema_name [, ...] }
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

#### CAUTION

Do not perform concurrent GRANT or REVOKE operations (including **ON TABLE** and **ON ALL TABLES IN SCHEMA**) on a given table or view, or do not perform both GRANT or REVOKE and DDL operations on a given table or view. Otherwise, an error similar to "tuple concurrently updated." is reported.

- Revoke the permission of specified fields on the table.

```
REVOKE [GRANT OPTION FOR]
 { { SELECT | INSERT | UPDATE | REFERENCES } (column_name [, ...])[, ...]
 | ALL [PRIVILEGES] (column_name [, ...]) }
 ON [TABLE] table_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the permission of a specified database.

```
REVOKE [GRANT OPTION FOR]
 { { CREATE | CONNECT | TEMPORARY | TEMP } [, ...]
 | ALL [PRIVILEGES] }
 ON DATABASE database_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the permission of a specified function.

```
REVOKE [GRANT OPTION FOR]
 { { EXECUTE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
 ON { FUNCTION {function_name ([{[argmode] [arg_name] arg_type} [, ...]) } } [, ...]
 | ALL FUNCTIONS IN SCHEMA schema_name [, ...] }
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the permission of a specified large object.

```
REVOKE [GRANT OPTION FOR]
 { { SELECT | UPDATE } [, ...] | ALL [PRIVILEGES] }
 ON LARGE OBJECT loid [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the permission on a specified sequence.

```
REVOKE [GRANT OPTION FOR]
 { { SELECT | UPDATE | USAGE } [, ...] | ALL [PRIVILEGES] }
 ON SEQUENCE sequence_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the permission of a specified schema.

```
REVOKE [GRANT OPTION FOR]
 { { CREATE | USAGE | ALTER | DROP } [, ...] | ALL [PRIVILEGES] }
 ON SCHEMA schema_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the permission of a specified sub-cluster.

```
REVOKE [GRANT OPTION FOR]
 { CREATE | USAGE | COMPUTE | ALL [PRIVILEGES] }
 ON NODE GROUP group_name [, ...]
 FROM { [GROUP] role_name | PUBLIC } [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the permission of roles based on roles.

```
REVOKE [ADMIN OPTION FOR]
 role_name [, ...] FROM role_name [, ...]
 [CASCADE | RESTRICT];
```

- Revoke the sysadmin permission of a specified role.

```
REVOKE ALL { PRIVILEGES | PRIVILEGE } FROM role_name;
```

## Parameter Description

The keyword **PUBLIC** indicates an implicitly defined group that has all roles. For details about the **REVOKE** permission and related parameters, see [Parameter Description](#) in "GRANT".



 **NOTE**

- Permissions of a role include the permissions directly granted to the role, permissions inherited from the parent role, and permissions granted to **PUBLIC**. Therefore, revoking the **SELECT** permission for an object from **PUBLIC** does not necessarily mean that the **SELECT** permission for the object has been revoked from all roles, because the **SELECT** permission directly granted to roles and inherited from parent roles still remains. Similarly, if the **SELECT** permission is revoked from a user and the **PUBLIC** user still has the **SELECT** permission, the user can still use the **SELECT** permission.
- If user A holds the **UPDATE** rights on a table and the **WITH GRANT OPTION** and has granted them to user B, the rights that user B holds are called dependent rights. If the rights or the grant option held by user A is revoked, the dependent rights still exist. Those dependent rights are also revoked if **CASCADE** is specified.
- A user can only revoke rights that were granted directly by that user. If, for example, user A has granted a right with grant option (**WITH ADMIN OPTION**) to user B, and user B has in turn granted it to user C, then user A cannot revoke the right directly from C. However, user A can revoke the grant option held by user B and use **CASCADE**. In this manner, the rights held by user C are automatically revoked. For another example, if both user A and user B have granted the same right to C, A can revoke his own grant but not B's grant, so C will still effectively have the right.
- If the role executing **REVOKE** holds rights indirectly via more than one role membership path, it is unspecified which containing role will be used to execute the command. In such cases, it is best practice to use **SET ROLE** to become the specific role you want to do the **REVOKE** as, and then execute **REVOKE**. Failure to do so may lead to deleting rights not intended to delete, or not deleting any rights at all.

## Examples

Revoke the sysadmin permission of user **joe**:

```
REVOKE ALL PRIVILEGES FROM joe;
```

Revoke the permissions granted in a specified schema.

```
REVOKE USAGE,CREATE ON SCHEMA tpcds FROM tpcds_manager;
```

Revoke the **CONNECT** privilege from user **joe**.

```
REVOKE CONNECT ON DATABASE mydatabase FROM joe;
```

Revoke the membership of role **admins** from user **joe**.

```
REVOKE admins FROM joe;
```

Revoke all the privileges of user **joe** for the **myView** view.

```
REVOKE ALL PRIVILEGES ON myView FROM joe;
```

Revoke the insert permission for the **customer\_t1** table from **public**.

```
REVOKE INSERT ON customer_t1 FROM PUBLIC;
```

Revoke user **joe**'s permission for the **tpcds** schema.

```
REVOKE USAGE ON SCHEMA tpcds FROM joe;
```

Revoke the query permissions for **r\_reason\_sk** and **r\_reason\_id** in the **tpcds.reason** table from user **joe**.

```
REVOKE select (r_reason_sk, r_reason_id) ON tpcds.reason FROM joe;
```

Revoke user **joe**'s execute permission on function **func\_add\_sql**.

```
REVOKE execute ON FUNCTION func_add_sql(integer, integer) FROM joe CASCADE;
```

## Links

[GRANT](#)

## 21.16 DQL Syntax

### 21.16.1 DQL Syntax Overview

Data Query Language (DQL) can obtain data from tables or views.

#### Query

DWS provides statements for obtaining data from tables or views. For details, see [SELECT](#).

#### Defining a New Table Based on Query Results

DWS provides a statement for creating a table based on query results and inserting the queried data into the table. For details, see [SELECT INTO](#).

### 21.16.2 SELECT

#### Function

Retrieves data from a table or view.

Serving as an overlaid filter for a database table, **SELECT** using SQL keywords retrieves required data from data tables.

#### Precautions

- For details about how to use SELECT, see "SELECT Operation" in *Data Warehouse Service (DWS) Developer Guide*.
- Using **SELECT** can join HDFS and ordinary tables, but cannot join ordinary and GDS foreign tables. That is, a **SELECT** statement cannot contain both ordinary and GDS foreign tables.
- You must have the SELECT permission on each column used in the **SELECT** command. If **FOR UPDATE** or **FOR SHARE** is used, the UPDATE permission is also required.

#### Syntax

The syntax format of the main clause is shown below. The syntax formats of the FROM and GROUP BY clauses are defined separately. For details about the FROM and GROUP BY clauses, see [FROM Clause Parameters](#) and [GROUP BY Parameters](#).

##### NOTE

SELECT supports the simplified query syntax, which is equivalent to **select \* from table\_name**.

```
TABLE { ONLY {(table_name)| table_name} | table_name [*]};
```

```
[WITH [RECURSIVE] with_query [, ...]]
SELECT [/*+ plan_hint */] [ALL | DISTINCT [ON (expression [, ...])]]
{ * | {expression [[AS] output_name]} [, ...] }
[FROM from_item [, ...]]
[WHERE condition]
[GROUP BY grouping_element [, ...]]
[HAVING condition [, ...]]
[WINDOW {window_name AS (window_definition)} [, ...]]
[{ UNION | INTERSECT | EXCEPT | MINUS } [ALL | DISTINCT] select]
[ORDER BY {expression [[ASC | DESC | USING operator] | nlssort_expression_clause } [NULLS { FIRST |
LAST }]} [, ...]]
[{ [LIMIT { count | ALL }] [OFFSET start [ROW | ROWS]] } | { LIMIT start, { count | ALL } }]
[FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY]
[{FOR { UPDATE | SHARE } [OF table_name [, ...]] [NOWAIT]} [...]];
```

---

#### NOTICE

In **condition** and **expression**, you can use the aliases of expressions in the output column in compliance with the following rules:

- Only aliases at the same level can be referenced.
  - Only aliases in the output column can be referenced.
  - A prior expression is referenced only in a subsequent expression.
  - The **volatile** function cannot be used.
  - The **Window** function cannot be used.
  - Do not reference an alias in the **join on** condition.
  - An error is reported if the output column contains multiple referenced aliases.
-

## SELECT Main Clause Parameters

Table 21-305 SELECT parameters

| Parameter                                      | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| WITH<br>[ RECURSIVE ]<br>with_query<br>[, ...] | <p>Specifies one or more subqueries (CTE) that can be referenced by name in the primary query. The subqueries act as temporary tables. If a given subquery result needs to be referenced multiple times, use WITH to simplify the code logic and improve the coding efficiency.</p> <p>If <b>RECURSIVE</b> is specified, it allows a <b>SELECT</b> subquery to reference itself by name.</p> <p>The format of <b>with_query</b>:</p> <pre>with_query_name [ ( column_name [, ...] ) ] AS<br/>[ [ NOT ] MATERIALIZED ] ( {select   values   insert  <br/>update   delete} )</pre> | <ul style="list-style-type: none"><li>• <b>with_query_name</b> specifies the name of the result set generated by the subquery. You can use this name to access the result set of the subquery.</li><li>• <b>[ NOT ] MATERIALIZED</b> specifies this parameter to change the default behavior.<ul style="list-style-type: none"><li>– If <b>MATERIALIZED</b> is specified, the subquery is executed once, and the result set is materialized for the primary query to query the result set for multiple times.</li><li>– If <b>NOT MATERIALIZED</b> is specified, <b>with_query</b> is referenced only once by the primary query and is not executed separately. Instead,</li></ul></li></ul> |

| Parameter | Description | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           |             | <p><b>with_query</b> is substituted with its reference in the primary query and executed alongside it.</p> <ul style="list-style-type: none"><li>- <b>NOT MATERIALIZED</b> is ignored in the following cases:<ul style="list-style-type: none"><li>- The subquery contains volatile functions.</li><li>- The subquery is a <b>SELECT</b> or <b>VALUES</b> statement containing <b>FOR UPDATE</b> or <b>FOR SHARE</b>.</li><li>- The subquery is an <b>INSERT</b>, <b>UPDATE</b>, or <b>DELETE</b> statement.</li><li>- <b>RECURSIVE</b> is specified for <b>with_query</b>.</li><li>- If <b>with_query2</b> is</li></ul></li></ul> |

| Parameter | Description | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           |             | <div>referenced more than once and it references <b>with_query1</b>, which referenced itself in the outer layer, <b>with_query2</b> cannot take the place where it can be referenced.</div> <div>For example, in the following example, <b>tmp2</b> is referenced twice. Because <b>tmp2</b> references <b>tmp1</b> which referenced itself in the outer layer, <b>tmp2</b> will be materialized even if <b>NOT MATERIALIZED</b> is specified.</div> <div>with recursive<br/>tmp1(b) as<br/>(values(1)<br/>union all</div> |

| Parameter                              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Value Range or Example                                                                                                                                                                                                                                                                                                                                                                                                               |
|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | <pre>(with tmp2<br/>as not<br/>materialized<br/>(select *<br/>from tmp1)<br/>select tt1.b<br/>+ tt2.b from<br/>tmp2 tt1,<br/>tmp2 tt2))<br/>select * from<br/>tmp1;</pre> <ul style="list-style-type: none"><li>• <b>column_name</b> specifies a column name displayed in the subquery result set.</li><li>• Each subquery can be a <b>SELECT</b>, <b>VALUES</b>, <b>INSERT</b>, <b>UPDATE</b> or <b>DELETE</b> statement.</li></ul> |
| <b>plan_hint</b> clause                | This clause, in the format <b>/*+ */</b> , comes after the <b>SELECT</b> keyword. It is used to enhance the plan of a <b>SELECT</b> statement block.<br><br>For details, see "DWS Performance Tuning" > "SQL Tuning" > "Hint-based Tuning" in the <i>Data Warehouse Service (DWS) Developer Guide</i> .                                                                                                                                                                               | -                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| ALL                                    | Specifies that all rows that meet the conditions are returned. This is the default behavior and can be omitted.                                                                                                                                                                                                                                                                                                                                                                       | -                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| DISTINCT [ ON ( expression [, ...] ) ] | Removes all duplicate rows from the <b>SELECT</b> result to ensure that each row is unique.<br><br><b>ON ( expression [, ...] )</b> is only reserved for the first row among all the rows with the same result calculated using given expressions.<br><br><b>NOTICE</b><br><b>DISTINCT ON</b> expression is explained with the same rule of <b>ORDER BY</b> . Unless you use <b>ORDER BY</b> to guarantee that the required row appears first, you cannot know what the first row is. | -                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| Parameter              | Description                                                                                                                                                                                                                                                                                                                  | Value Range or Example                                                                                                                                                                                                                                                                                                                   |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SELECT</b> list     | Indicates columns to be queried. Some or all columns (using wildcard character *) can be queried.<br><br>You may use the <b>AS output_name</b> clause to give an alias for an output column. The alias is used for the displaying of the output column.                                                                      | Column names may be either of: <ul style="list-style-type: none"><li>Manually input column names which are spaced using commas (,).</li><li>Fields computed in the <b>FROM</b> clause.</li><li>Generally, the keyword <b>AS</b> can be omitted. For example, <b>select 2a</b> is implicitly converted to <b>select 2 as a</b>.</li></ul> |
| <b>FROM</b> clause     | Indicates one or more source tables for <b>SELECT</b> . For details, see <a href="#">Table 21-306</a> .                                                                                                                                                                                                                      | -                                                                                                                                                                                                                                                                                                                                        |
| <b>WHERE</b> clause    | The <b>WHERE</b> clause forms an expression for row selection to narrow down the query range of <b>SELECT</b> .                                                                                                                                                                                                              | <ul style="list-style-type: none"><li><b>LIKE</b> operator</li><li>The <b>(+)</b> operator is not recommended.</li></ul> For details, see <a href="#">Restrictions on the WHERE Clause</a> .                                                                                                                                             |
| <b>GROUP BY</b> clause | Condenses query results into a single row or selected rows that share the same values for the grouped expressions.                                                                                                                                                                                                           | For details, see <a href="#">Table 21-308</a> .                                                                                                                                                                                                                                                                                          |
| <b>HAVING</b> clause   | Filters group rows created by <b>GROUP BY</b> . It is similar to <b>WHERE</b> , but is usually used with <b>GROUP BY</b> to filter out special groups. The <b>HAVING</b> clause compares some attributes of groups with a constant. Only groups that match the logical expression in the <b>HAVING</b> clause are extracted. | -                                                                                                                                                                                                                                                                                                                                        |



| Parameter               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Value Range or Example |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <b>WINDOW</b><br>clause | <p>Defines a window function. The format is <b>WINDOW</b> <i>window_name</i> <b>AS</b> ( <i>window_definition</i> ) [, ...]. <i>window_name</i> indicates a name that can be referenced from subsequent window definitions. <i>window_definition</i> indicates the window rule, which can be expressed in the following format:</p> <pre>[ existing_window_name ]<br/>[ PARTITION BY expression [, ...] ]<br/>[ ORDER BY expression [ ASC   DESC   USING<br/>operator ] [ NULLS { FIRST   LAST } ] [, ...] ]<br/>[ frame_clause ]</pre> <p><b>frame_clause</b> defines a <b>window frame</b> for the window function. The window function (not all window functions) depends on <b>window frame</b> and <b>window frame</b> is a set of relevant rows of the current query row. <b>frame_clause</b> can be expressed in the following forms:</p> <pre>[ RANGE   ROWS ] frame_start<br/>[ RANGE   ROWS ] BETWEEN frame_start AND<br/>frame_end</pre> <p><b>frame_start</b> and <b>frame_end</b> can be expressed in the following forms:</p> <pre>UNBOUNDED PRECEDING<br/>value PRECEDING (not supported for <b>RANGE</b>)<br/>CURRENT ROW<br/>value FOLLOWING (not supported for <b>RANGE</b>)<br/>UNBOUNDED FOLLOWING</pre> <p><b>NOTICE</b><br/>For the query of column storage table, only <b>row_number</b> window function is supported, <b>frame_clause</b> is not supported.</p> | -                      |

| Parameter               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Value Range or Example |
|-------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <b>UNION</b> clause     | <p>Computes the set union of the rows returned by the involved <b>SELECT</b> statements.</p> <p>The <b>UNION</b> clause has the following constraints:</p> <ul style="list-style-type: none"><li>• By default, the result of <b>UNION</b> does not contain any duplicate rows unless the <b>ALL</b> option is specified.</li><li>• Multiple <b>UNION</b> operators in the same <b>SELECT</b> statement are evaluated left to right, unless otherwise specified by parentheses.</li><li>• <b>FOR UPDATE</b> cannot be specified either for a <b>UNION</b> result or for any input of a <b>UNION</b>.</li></ul> <p>General expression:</p> <pre>select_statement UNION [ALL] select_statement</pre> <ul style="list-style-type: none"><li>• <b>select_statement</b> can be any <b>SELECT</b> statement without an <b>ORDER BY</b>, <b>LIMIT</b>, or <b>FOR UPDATE</b> clause.</li><li>• <b>ORDER BY</b> and <b>LIMIT</b> in parentheses can be attached in a sub-expression.</li></ul> | -                      |
| <b>INTERSECT</b> clause | <p>Computes the set intersection of rows returned by the involved <b>SELECT</b> statements. The result of <b>INTERSECT</b> does not contain any duplicate rows.</p> <p>The <b>INTERSECT</b> clause has the following constraints:</p> <ul style="list-style-type: none"><li>• Multiple <b>INTERSECT</b> operators in the same <b>SELECT</b> statement are evaluated left to right, unless otherwise specified by parentheses.</li><li>• Processing <b>INTERSECT</b> preferentially when <b>UNION</b> and <b>INTERSECT</b> operations are executed for results of multiple <b>SELECT</b> statements.</li></ul> <p>General format:</p> <pre>select_statement INTERSECT select_statement</pre> <p><b>select_statement</b> can be any <b>SELECT</b> statement without a <b>FOR UPDATE</b> clause.</p>                                                                                                                                                                                    | -                      |

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Value Range or Example |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <b>EXCEPT</b> clause | <p>The <b>EXCEPT</b> operator computes the set of rows that are in the result of the left <b>SELECT</b> statement but not in the result of the right one.</p> <p><b>EXCEPT</b> clause has the following common form:</p> <pre>select_statement EXCEPT [ ALL ] select_statement</pre> <ul style="list-style-type: none"><li>• <b>select_statement</b> can be any <b>SELECT</b> statement without a <b>FOR UPDATE</b> clause.</li><li>• The result of <b>EXCEPT</b> does not contain any duplicate rows unless the <b>ALL</b> option is specified. To execute <b>ALL</b>, a row that has <math>m</math> duplicates in the left table and <math>n</math> duplicates in the right table will appear <math>\text{MAX}(m-n, 0)</math> times in the result set.</li><li>• Multiple <b>EXCEPT</b> operators in the same <b>SELECT</b> statement are evaluated left to right, unless parentheses dictate otherwise. <b>EXCEPT</b> binds at the same level as <b>UNION</b>.</li><li>• Currently, <b>FOR UPDATE</b> cannot be specified either for an <b>EXCEPT</b> result or for any input of an <b>EXCEPT</b>.</li></ul> | -                      |
| <b>MINUS</b> clause  | Has the same function and syntax as <b>EXCEPT</b> clause.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | -                      |

| Parameter                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Value Range or Example                                                                                                                |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <b>ORDER BY</b> clause    | <p>Sorts data retrieved by <b>SELECT</b> in descending or ascending order. If the <b>ORDER BY</b> expression contains multiple columns:</p> <ul style="list-style-type: none"><li>• If two columns are equal according to the leftmost expression, they are compared according to the next expression and so on.</li><li>• If they are equal according to all specified expressions, they are returned in an implementation-dependent order.</li><li>• Columns sorted by <b>ORDER BY</b> must be contained in the result retrieved by <b>SELECT</b>.</li></ul> <p><b>NOTICE</b></p> <ul style="list-style-type: none"><li>• If <b>ORDER BY</b> is not specified, the query results are returned following the generation sequence in the database system.</li><li>• You can add the keyword <b>ASC</b> (in ascending order) or <b>DESC</b> (in descending order) next to any expression in the <b>ORDER BY</b> clause. If the keyword is not specified, <b>ASC</b> is used by default.</li><li>• To sort query results by case-insensitive Chinese pinyin, set the encoding mode to <b>UTF-8</b> or <b>GBK</b> during database creation.</li></ul> | -                                                                                                                                     |
| nlssort_expression_clause | <p>Sets the sorting mode. The format is as follows:</p> <pre>NLSSORT ( column_name, ' NLS_SORT = { SCHINESE_PINYIN_M   generic_m_ci } ' )</pre> <p><b>NLS_SORT</b>: indicates a field to be ordered in a special mode. Currently, only the Chinese Pinyin order and case insensitive order are supported. Values:</p> <ul style="list-style-type: none"><li>• <b>SCHINESE_PINYIN_M</b>: Chinese characters are sorted by pinyin. Currently, only level-1 Chinese characters in the GBK character set can be sorted. To use this sort method, specify <b>GBK</b> as the encoding format when you create the database. If you do not do so, this value is invalid.</li><li>• <b>generic_m_ci</b>, case-insensitive order.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                  | <ul style="list-style-type: none"><li>• <a href="#">Example: Use NLSSORT to Sort Query Results Without Case Sensitivity</a></li></ul> |

| Parameter                                                                                                                    | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Value Range or Example                                                |
|------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------|
| [ { [ LIMIT<br>{ count  <br>ALL } ]<br>[ OFFSET start<br>[ ROW  <br>ROWS ] ] }  <br>{ LIMIT start,<br>{ count  <br>ALL } } ] | <p>The <b>LIMIT</b> clause consists of two independent <b>LIMIT</b> clauses, an <b>OFFSET</b> clause, and a <b>LIMIT</b> clause with multiple parameters.</p> <p>LIMIT { count   ALL }<br/>OFFSET start [ ROW   ROWS ]<br/>LIMIT start, { count   ALL }</p> <p><b>count</b> in the clauses specifies the maximum number of rows to return, while <b>start</b> specifies the number of rows to skip before starting to return rows. When both are specified, <b>start</b> rows are skipped before starting to count the <b>count</b> rows to be returned. A multi-parameter <b>LIMIT</b> clause cannot be used together with a single-parameter <b>LIMIT</b> or <b>OFFSET</b> clause.</p> <p>Example:<br/>LIMIT 10 OFFSET 20; -- Skip the first 20 rows and return the next 10 rows, that is, return rows 21 to 30.<br/>LIMIT 5, 10; -- Skip the first five lines and returns the last 10 rows, that is, returns the rows 6 to 16.</p> | -                                                                     |
| FETCH { FIRST  <br>NEXT }<br>[ count ]<br>{ ROW  <br>ROWS } ONLY                                                             | The <b>FETCH</b> clause restricts the total number of rows starting from the first row of the return query result.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | If <b>count</b> is not specified, the default value <b>1</b> is used. |

| Parameter                 | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Value Range or Example |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| FOR UPDATE   SHARE clause | <p>The <b>FOR UPDATE</b> clause locks the selected rows in a <b>SELECT</b> query to prevent other transactions from modifying or deleting them until the current transaction is completed. This means that any transactions attempting to execute <b>UPDATE</b>, <b>DELETE</b>, or <b>SELECT FOR UPDATE</b> on these rows will be blocked until the current transaction is finished.</p> <p><b>NOWAIT</b> option: prevents an operation from waiting for other transactions to commit. If the selected row cannot be locked immediately, an error is reported when <b>SELECT FOR UPDATE NOWAIT</b> is executed.</p> <ul style="list-style-type: none"><li>• If specified tables are named in <b>FOR UPDATE/SHARE</b>, then only rows coming from those tables are locked; any other tables used in <b>SELECT</b> are simply read as usual. Otherwise, locking all tables in the command.</li><li>• If <b>FOR UPDATE</b> or <b>FOR SHARE</b> is applied to a view or sub-query, it affects all tables used in the view or sub-query.</li><li>• Multiple <b>FOR UPDATE</b> and <b>FOR SHARE</b> clauses can be written if it is necessary to specify different locking behaviors for different tables.</li><li>• If the same table is mentioned (or implicitly affected) by both <b>FOR UPDATE</b> and <b>FOR SHARE</b> clauses, it is processed as <b>FOR UPDATE</b>. Similarly, a table is processed as <b>NOWAIT</b> if that is specified in any of the clauses affecting it.</li><li>• <b>FOR SHARE</b> or <b>FOR UPDATE</b>: requires a shared lock instead of an exclusive lock for each retrieved row. A share lock blocks other transaction from performing <b>UPDATE</b>, <b>DELETE</b>, or <b>SELECT FOR UPDATE</b> on these rows, but it does not prevent them from performing <b>SELECT FOR SHARE</b>.</li></ul> | -                      |

| Parameter | Description                                                                                                                                                                                                                                                                                                                             | Value Range or Example |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
|           | <b>NOTICE</b> <ul style="list-style-type: none"><li>For SQL statements containing <b>FOR UPDATE</b> or <b>FOR SHARE</b>, their execution plans will be pushed down to DN. If the pushdown fails, an error will be reported.</li><li>The query of column storage table does not support <b>FOR UPDATE</b> or <b>FOR SHARE</b>.</li></ul> |                        |

## FROM Clause Parameters

FROM from\_item clause format:

```
{[ONLY] table_name [*] [partition_clause] [[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 [, ...])]] }
```

**Table 21-306** FROM clause parameters

| Parameter    | Description                                                                                                                                                                                                                                             | Value Range or Example                                                                                                          |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| table_name   | Indicates the name (optionally schema-qualified) of an existing table or view, for example, <b>schema_name.table_name</b> .                                                                                                                             | A string, which must comply with the naming convention.<br><br>For details, see <a href="#">Identifier Naming Conventions</a> . |
| alias        | Gives a temporary alias to a table to facilitate the quotation by other queries.<br><br>An alias is used for brevity or to eliminate ambiguity for self-joins. When an alias is provided, it completely hides the actual name of the table or function. | A string, which must comply with the naming convention.<br><br>For details, see <a href="#">Identifier Naming Conventions</a> . |
| column_alias | Specifies the column alias.                                                                                                                                                                                                                             | A string, which must comply with the naming convention.<br><br>For details, see <a href="#">Identifier Naming Conventions</a> . |

| Parameter        | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Value Range or Example                                                                                                                                                                                                                                                                                                                                      |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| partition_clause | <p>Queries data in the specified partition of a partitioned table. The format is as follows:</p> <pre>PARTITION { ( partition_name )  <br/>FOR ( partition_value [, ...] ) }</pre> <ul style="list-style-type: none"><li>• <b>partition_name</b>: indicates the partition name.</li><li>• <b>partition_value</b>: indicates the partition key value. If there are many partition keys, use the <b>PARTITION FOR</b> clause to specify the value of the only partition key you want to use.</li></ul> <p><b>NOTICE</b><br/>Partitions can be specified only for ordinary tables.</p> | <p>Examples:</p> <pre>SELECT * FROM<br/>tpcds.reason_p<br/>PARTITION<br/>(P_05_BEFORE);</pre>                                                                                                                                                                                                                                                               |
| subquery         | <p>Performs a subquery in the <b>FROM</b> clause. A temporary table is created to save subquery results.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | <p>For example, assign an alias <b>e</b> to a subquery result and use it as a <b>FROM</b> data source.</p> <pre>SELECT<br/>d.department_name,<br/>e.avg_salary FROM<br/><br/>( SELECT department_id,<br/>AVG(salary) AS<br/>avg_salary FROM<br/>employees GROUP<br/>BY department_id) AS e<br/><br/>JOIN departments d ON<br/>e.department_id = d.id;</pre> |
| with_query_name  | <p><b>WITH</b> clause can also be the source of <b>FROM</b> clause and can be referenced with the name queried by executing <b>WITH</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                        | <p>A string, which must comply with the naming convention.</p> <p>For details, see <a href="#">Identifier Naming Conventions</a>.</p> <p>For example, use <b>WITH cte</b> and reference <b>cte</b> in a <b>FROM</b> clause.</p> <pre>WITH cte AS (SELECT id,<br/>name FROM users)<br/>SELECT * FROM cte AS<br/>user_info;</pre>                             |



| Parameter                 | Description                                                                                                                       | Value Range or Example                                                                                                                                                                                                                                 |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| function_name             | Specifies the function name. Function calls can appear in the <b>FROM</b> clause.                                                 | A string, which must comply with the naming convention.<br>For details, see <a href="#">Identifier Naming Conventions</a> .                                                                                                                            |
| join_type                 | Specifies the JOIN type.                                                                                                          | Supported JOIN types: <ul style="list-style-type: none"><li>• [ INNER ] JOIN</li><li>• LEFT [ OUTER ] JOIN</li><li>• RIGHT [ OUTER ] JOIN</li><li>• FULL [ OUTER ] JOIN</li><li>• CROSS JOIN</li></ul> For details, see <a href="#">Table 21-307</a> . |
| ON join_condition         | A join condition to define which rows have matches in joins. Example: ON left_table.a = right_table.a                             | -                                                                                                                                                                                                                                                      |
| USING(join_column[, ...]) | ON left_table.a = right_table.a AND left_table.b = right_table.b ... abbreviation. Corresponding columns must have the same name. | -                                                                                                                                                                                                                                                      |
| NATURAL                   | <b>NATURAL</b> is a shorthand for a <b>USING</b> list that mentions all columns in the two tables that have the same names.       | -                                                                                                                                                                                                                                                      |
| from item                 | Specifies the name of the query source object connected.                                                                          | -                                                                                                                                                                                                                                                      |

**Table 21-307** Five types of join\_type

| Parameter      | Description                                                                                                                                                                                                                                                                       |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [ INNER ] JOIN | A <b>JOIN</b> clause consists of two <b>FROM</b> items. Use parentheses if necessary to determine the order of nesting. In the absence of parentheses, <b>JOIN</b> nests left-to-right. In any case, <b>JOIN</b> binds more tightly than the commas separating <b>FROM</b> items. |

| Parameter            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LEFT [ OUTER ] JOIN  | Returns all rows in the qualified Cartesian product (all combined rows that pass its join condition), and pluses one copy of each row in the left-hand table for which there was no right-hand row that passed the join condition. This left-hand row is extended to the full width of the joined table by inserting <b>NULL</b> values for the right-hand columns.<br><br>Note that only the <b>JOIN</b> clause's own condition is considered while deciding which rows have matches. Outer conditions are applied afterwards.                                                                                                                                                                                                                         |
| RIGHT [ OUTER ] JOIN | Returns all the joined rows, plus one row for each unmatched right-hand row (extended with <b>NULL</b> on the left). This is just a notational convenience, since you could convert it to a <b>LEFT OUTER JOIN</b> by switching the left and right inputs.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| FULL [ OUTER ] JOIN  | Returns all the joined rows, pluses one row for each unmatched left-hand row (extended with <b>NULL</b> on the right), and pluses one row for each unmatched right-hand row (extended with <b>NULL</b> on the left).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| CROSS JOIN           | <b>CROSS JOIN</b> is equivalent to <b>INNER JOIN ON (TRUE)</b> , which means no rows are removed by qualification. These join types are just a notational convenience, since they do nothing you could not do with plain <b>FROM</b> and <b>WHERE</b> .<br><b>NOTE</b> <ul style="list-style-type: none"><li>For the <b>INNER</b> and <b>OUTER</b> join types, a join condition must be specified, namely exactly one of <b>NATURAL ON</b>, <b>join_condition</b>, or <b>USING (join_column [, ...])</b>. For <b>CROSS JOIN</b>, none of these clauses can appear.</li><li><b>CROSS JOIN</b> and <b>INNER JOIN</b> produce a simple Cartesian product, the same result as you get from listing the two items at the top level of <b>FROM</b>.</li></ul> |

## GROUP BY Parameters

The **group** clause is as follows:

```
(
| expression
| (expression [, ...])
| ROLLUP ({ expression | (expression [, ...]) } [, ...])
| CUBE ({ expression | (expression [, ...]) } [, ...])
| GROUPING SETS (grouping_element [, ...])
```

**Table 21-308** GROUP By parameters

| Parameter                                                              | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Examples |
|------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| ROLLUP<br>( { expression  <br>( expression<br>[, ...] ) }<br>[, ...] ) | <p>ROLLUP calculates the standard aggregation value specified by an ordered grouping column in GROUP BY, creates a high-level partial sum from right to left, and finally creates a cumulative sum. A group can be regarded as a series of grouping sets. Example:<br/>GROUP BY ROLLUP (a,b,c)</p> <p>The preceding condition expression is equivalent to:<br/>GROUP BY GROUPING SETS((a,b,c), (a,b), (a), ( ))</p> <p>The elements in the <b>ROLLUP</b> clause can be independent fields or expressions, or a list contained in parentheses. If it is a list in parentheses, they must be a whole when the grouping set is generated. Example:<br/>GROUP BY ROLLUP ((a,b), (c,d))</p> <p>The preceding condition expression is equivalent to:<br/>GROUPING SETS ((a,b,c,d), (a,b), (c,d), ( ))</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Examples |
| CUBE<br>( { expression  <br>( expression<br>[, ...] ) }<br>[, ...] )   | <p>A CUBE grouping is an extension to the GROUP BY clause that creates subtotals for all of the possible combinations of the given list of grouping columns (or expressions). CUBE can be understood as all dimensions, that is, all combinations of columns in the aggregation key.</p> <p>In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions. For example, given three expressions (<math>n=3</math>) in the CUBE clause, the operation results in <math>2^n = 2^3 = 8</math> groupings. Rows grouped on the values of <math>n</math> expressions are called regular rows, and the rest are called superaggregate rows. Example:<br/>GROUP BY CUBE (a,b,c)</p> <p>The preceding condition expression is equivalent to:<br/>GROUP BY GROUPING SETS((a,b,c), (a,b), (a,c), (b,c), (a), (b), (c), ( ))</p> <p>The elements in the <b>CUBE</b> clause can be independent fields or expressions, or a list contained in parentheses. If it is a list in parentheses, they must be a whole when the grouping set is generated. Example:<br/>GROUP BY CUBE (a, (b, c), d)</p> <p>The preceding condition expression is equivalent to:<br/>GROUP BY GROUPING SETS ((a,b,c,d), (a,b,c), (a), ( ))</p> | Examples |

| Parameter                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Examples                 |
|-----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| GROUPING SETS<br>( grouping_element [, ...] ) | <p><b>GROUPING SETS</b> is another extension to the <b>GROUP BY</b> clause. It allows users to specify multiple <b>GROUP BY</b> clauses. The option is used to define a grouping set. Each grouping set needs to be included in a separate parenthesis. A blank parenthesis (()) indicates that all data is processed as a group. This improves efficiency by trimming away unnecessary data. You can specify the required data group for query.</p> <p><b>NOTICE</b><br/>If the <b>SELECT</b> list expression quotes some ungrouped fields and no aggregate function is used, an error is displayed. This is because multiple values may be returned for ungrouped fields.</p> | <a href="#">Examples</a> |

## Restrictions on the WHERE Clause

1. For the **WHERE** clause, if a special character %, \_, or \ is queried in the **LIKE** operator, use the backslash (\) for escape.

```
CREATE TABLE tt01 (id int,content varchar(50));

INSERT INTO tt01 values (1,'Jack say "hello"');
INSERT INTO tt01 values (2,'Rose do 50%');
INSERT INTO tt01 values (3,'Lilei say "world"');
INSERT INTO tt01 values (4,'Hanmei do 100%');

SELECT * FROM tt01;
id | content
---+-----
3 | Lilei say 'world'
4 | Hanmei do 100%
1 | Jack say 'hello'
2 | Rose do 50%
(4 rows)

SELECT * FROM tt01 WHERE content like '%"he%';
id | content
---+-----
1 | Jack say 'hello'
(1 row)

SELECT * FROM tt01 WHERE content like '%50\%';
id | content
---+-----
2 | Rose do 50%
(1 row)
```

2. In the **WHERE** clause, you can use the operator (+) to convert a table join to an outer join. However, this method is not recommended because it is not the standard SQL syntax and may raise syntax compatibility issues during platform migration. The restrictions on using the plus sign (+) are as follows:
  - a. It can appear only in the **WHERE** clause.
  - b. If a table join has been specified in the **FROM** clause, the operator (+) cannot be used in the **WHERE** clause.
  - c. The operator (+) can work only on columns of tables or views, instead of on expressions.

- d. If table A and table B have multiple join conditions, the operator (+) must be specified in all the conditions. Otherwise, the operator (+) will not take effect, and the table join will be converted into an inner join without any prompt information.
- e. Tables specified in a join condition where the operator (+) works cannot cross queries or subqueries. If tables where the operator (+) works are not in the **FROM** clause of the current query or subquery, an error will be reported. If a peer table for the operator (+) does not exist, no error will be reported and the table join will be converted into an inner join.
- f. Expressions where the operator (+) is used cannot be directly connected through **OR**.
- g. If a column where the operator (+) works is compared with a constant, the expression becomes a part of the join condition.
- h. A table cannot have multiple foreign tables.
- i. The operator (+) can appear only in the following expressions: comparison, **NOT**, **ANY**, **ALL**, **IN**, **NULLIF**, **IS DISTINCT FROM**, and **IS OF** expressions. It is not allowed in other types of expressions. In addition, **AND** and **OR** are not allowed in these expressions.
- j. The operator (+) can be used to convert a table join only to a left or right outer join, instead of a full join. That is, the operator (+) cannot be specified on both tables of an expression.

### Example: Use WITH to Implement Subqueries

Obtain the **temp\_t** temporary table by a subquery and query all records in this table.

```
WITH temp_t(name,isdba) AS (SELECT username,usesuper FROM pg_user) SELECT * FROM temp_t;
```

Explicitly specify **MATERIALIZED** for the **with\_query** named **temp\_t**, and then query all data in the **temp\_t** table.

```
WITH temp_t(name,isdba) AS MATERIALIZED (SELECT username,usesuper FROM pg_user) SELECT * FROM temp_t;
```

Explicitly specify **NOT MATERIALIZED** for the **with\_query** named **temp\_t**, and then query all data in the **temp\_t** table.

```
WITH temp_t(name,isdba) AS NOT MATERIALIZED (SELECT username,usesuper FROM pg_user)
SELECT * FROM temp_t t1 WHERE name LIKE 'A%'
UNION ALL
SELECT * FROM temp_t t2 WHERE name LIKE 'B%';
```

### Example: Other Common SELECT Scenarios

Query all the **r\_reason\_sk** records in the **tpcds.reason** table and de-duplicate them.

```
SELECT DISTINCT(r_reason_sk) FROM tpcds.reason;
```

Example of a **LIMIT** clause: Obtain a record from the table.

```
SELECT * FROM tpcds.reason LIMIT 1;
```

Example of a **LIMIT** clause: Obtain the third record from the table.

```
SELECT * FROM tpcds.reason LIMIT 1 OFFSET 2;
```

Example of a **LIMIT** clause: Obtain the first two records from a table.

```
SELECT * FROM tpcds.reason LIMIT 2;
```

Query all records and sort them in alphabetic order.

```
SELECT r_reason_desc FROM tpcds.reason ORDER BY r_reason_desc;
```

Use table aliases to obtain data from the **pg\_user** and **pg\_user\_status** tables.

```
SELECT a.username,b.locktime FROM pg_user a,pg_user_status b WHERE a.usesysid=b.roloid;
```

Example of the **FULL JOIN** clause: Join data in the **pg\_user** and **pg\_user\_status** tables.

```
SELECT a.username,b.locktime,a.usesuper FROM pg_user a FULL JOIN pg_user_status b on a.usesysid=b.roloid;
```

Example of the **GROUP BY** clause: Filter data based on query conditions, and group the results.

```
SELECT r_reason_id, AVG(r_reason_sk) FROM tpcds.reason GROUP BY r_reason_id HAVING AVG(r_reason_sk) > 25;
```

Example of the **GROUP BY** clause: Group the results by alias.

```
SELECT r_reason_id AS id FROM tpcds.reason GROUP BY id;
```

Example of the **UNION** clause: Merge the names started with W and N in the **r\_reason\_desc** column in the **tpcds.reason** table.

```
SELECT r_reason_sk, tpcds.reason.r_reason_desc
FROM tpcds.reason
WHERE tpcds.reason.r_reason_desc LIKE 'W%'
UNION
SELECT r_reason_sk, tpcds.reason.r_reason_desc
FROM tpcds.reason
WHERE tpcds.reason.r_reason_desc LIKE 'N%';
```

## Example: Use WITH RECURSIVE to Query the Hierarchical Relationship of a Three-Level Department in a Company

Take the recursion of three levels of departments in a company as an example. In a table that contains level-1, level-2, and level-3 departments, use the keyword **RECURSIVE** to query all level-2 and level-3 departments of a department (for example, the technology department).

Prepare data:

```
DROP TABLE IF EXISTS departments;
CREATE TABLE departments
(id varchar(10),
pid varchar(10),
dept_name varchar(50));
INSERT INTO departments VALUES('0',null,'Head office');
INSERT INTO departments VALUES('1','0','Technology'); -- Level-1 department
INSERT INTO departments VALUES('2','0','Marketing');
INSERT INTO departments VALUES('3','0','HR');
INSERT INTO departments VALUES('4','0','Business');
INSERT INTO departments VALUES('5','0','Maintenance');
INSERT INTO departments VALUES('1-1','1','R&D Center'); -- Level-2 department
INSERT INTO departments VALUES('1-2','1','Pre-Research');
INSERT INTO departments VALUES('1-3','1','Network');
INSERT INTO departments VALUES('1-4','1','Test');
INSERT INTO departments VALUES('1-5','1','Architecture');
```

```
INSERT INTO departments VALUES('1-1-1','1-1','Frontend'); -- Level-3 department
INSERT INTO departments VALUES('1-1-2','1-1','Backend');
INSERT INTO departments VALUES('1-1-3','1-1','Design');
INSERT INTO departments VALUES('2-1','2','Branding');
INSERT INTO departments VALUES('2-2','2','Pricing');
INSERT INTO departments VALUES('2-3','2','Training');
```

Query the original data of the table. In the command output, it is difficult to determine the relationship of the three levels of departments.

```
SELECT * FROM departments;
```

| id    | pid    | dept_name    |
|-------|--------|--------------|
| 1     | 0      | Technology   |
| 4     | 0      | Business     |
| 1-2   | 1      | Pre-Research |
| 1-5   | 1      | Architecture |
| 1-1-3 | 1-1    | Design       |
| 2-3   | 2      | Training     |
| 0     | (Null) | Head office  |
| 3     | 0      | HR           |

In this case, use the keyword **WITH RECURSIVE** to query all level-2 and level-3 departments under the technology department.

```
WITH RECURSIVE t1 AS (-- Define a common expression named t1.
SELECT id,pid,dept_name FROM departments WHERE id='1' -- Initialize the recursive root node: Query the
record whose ID is 1, that is, the technology department.
UNION ALL --Recursively combine results (duplicate items are retained).
SELECT --Recursive part: query data layer by layer.
t2.id, --Subnode ID, that is, the name of each level-1 department.
t2.pid, --Parent node ID, that is, the head office.
t2.dept_name --Subnode name.
FROM departments t2
INNER JOIN t1 ON t2.pid = t1.id --Connect the recursive result through the parent node ID.
)
SELECT id,dept_name FROM t1 ORDER BY id; --(Final query) All level-2 and level-3 departments are
displayed by ID.
```

| id    | dept_name    |
|-------|--------------|
| 1     | Technology   |
| 1-1   | R&D Center   |
| 1-1-1 | Frontend     |
| 1-1-2 | Backend      |
| 1-1-3 | Design       |
| 1-2   | Pre-Research |
| 1-3   | Network      |
| 1-4   | Test         |
| 1-5   | Architecture |

View that the preceding query results are not intuitive. To display the department of each level, run the following command:

```
WITH RECURSIVE t1 AS (
 SELECT --Initialize the root node of recursion: Query the record whose ID is 1, that is, the technology
 department, and convert data type from record to text.
 id,
 pid,
 dept_name::text -- Explicitly convert the data type to text (prepared for subsequent path concatenation).
 FROM departments
 WHERE id='1'

 UNION ALL

 SELECT --Recursive part: concatenate the level path.
 t2.id,
 t2.pid,
 (t1.dept_name||'-'>||t2.dept_name)::text AS dept_name --Concatenate the parent path and the current
 node name, for example, Technology -> R&D Center.
 FROM departments t2
 INNER JOIN t1 ON t2.pid = t1.id)

SELECT id,dept_name FROM t1 ORDER BY id; --(Final query) Display the result with the level path
```



| id    | dept_name                        |
|-------|----------------------------------|
| 1     | Technology                       |
| 1-1   | Technology->R&D Center           |
| 1-1-1 | Technology->R&D Center->Frontend |
| 1-1-2 | Technology->R&D Center->Backend  |
| 1-1-3 | Technology->R&D Center->Design   |
| 1-2   | Technology->Pre-Research         |
| 1-3   | Technology->Network              |
| 1-4   | Technology->Test                 |
| 1-5   | Technology->Architecture         |

## Example: Use NLSSORT to Sort Query Results Without Case Sensitivity

Case-insensitive order:

```
CREATE TABLE stu_icode_info (id bigint, name text) DISTRIBUTE BY REPLICATION;
INSERT INTO stu_icode_info VALUES (1, 'aaaa'), (2, 'AAAA');
SELECT * FROM stu_icode_info ORDER BY NLSSORT (name, 'NLS_SORT = generic_m_ci');
id | name
----+-----
1 | aaaa
2 | AAAA
(2 rows)
```

## Example: Query Data in a Partitioned Table

Create the partitioned table **tpcds.reason\_p**, insert data, and obtain data from the **P\_05\_BEFORE** partition of the table.

```
CREATE TABLE tpcds.reason_p
(
 r_reason_sk integer,
 r_reason_id character(16),
 r_reason_desc character(100)
)
PARTITION BY RANGE (r_reason_sk)
(
 partition P_05_BEFORE values less than (05),
 partition P_15 values less than (15),
 partition P_25 values less than (25),
 partition P_35 values less than (35),
 partition P_45_AFTER values less than (MAXVALUE)
);

INSERT INTO tpcds.reason_p values(3,'AAAAAAAAABAAAAAAAA','reason 1'),
(10,'AAAAAAAAABAAAAAAAA','reason 2'),(4,'AAAAAAAAABAAAAAAAA','reason 3'),
(10,'AAAAAAAAABAAAAAAAA','reason 4'),(10,'AAAAAAAAABAAAAAAAA','reason 5'),
```

```
(20,'AAAAAAAACAAAAAA','reason 6'),(30,'AAAAAAAACAAAAAA','reason 7');
```

```
SELECT * FROM tpcds.reason_p PARTITION (P_05_BEFORE);
r_reason_sk | r_reason_id | r_reason_desc
```

```
-----+-----+-----
 4 | AAAAAAABAAAAAA | reason 3
 3 | AAAAAAABAAAAAA | reason 1
```

```
(2 rows)
```

—Query the number of rows in partition **P\_15**:

```
SELECT count(*) FROM tpcds.reason_p PARTITION (P_15);
count
```

```

 3
(1 row)
```

Example of the **GROUP BY** clause: Group records in the **tpcds.reason\_p** table by **r\_reason\_id**, and count the number of records in each group.

```
SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id;
count | r_reason_id
```

```
-----+-----
 2 | AAAAAAABAAAAAA
 5 | AAAAAAABAAAAAA
```

```
(2 rows)
```

Example of the **HAVING** clause: Group records in the **tpcds.reason\_p** table by **r\_reason\_id**, count the number of records in each group, and display only values whose number of **r\_reason\_id** is greater than 2.

```
SELECT COUNT(*) c,r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING c>2;
c | r_reason_id
```

```
-----+-----
 5 | AAAAAAABAAAAAA
```

```
(1 row)
```

Example of the **IN** clause: Group records in the **tpcds.reason\_p** table by **r\_reason\_id**, count the number of records in each group, and display only the numbers of records whose **r\_reason\_id** is **AAAAAAAABAAAAAA** or **AAAAAAAADAAAAAA**.

```
SELECT COUNT(*),r_reason_id FROM tpcds.reason_p GROUP BY r_reason_id HAVING r_reason_id
IN('AAAAAAAABAAAAAA','AAAAAAAADAAAAAA');
count | r_reason_id
```

```
-----+-----
 5 | AAAAAAABAAAAAA
```

```
(1 row)
```

Example of the **INTERSECT** clause: Query records whose **r\_reason\_id** is **AAAAAAAABAAAAAA** and whose **r\_reason\_sk** is smaller than 5.

```
SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAA' INTERSECT SELECT * FROM
tpcds.reason_p WHERE r_reason_sk<5;
```

```
r_reason_sk | r_reason_id | r_reason_desc
```

```
-----+-----+-----
 4 | AAAAAAABAAAAAA | reason 3
 3 | AAAAAAABAAAAAA | reason 1
```

```
(2 rows)
```

Example of the **EXCEPT** clause: Query records whose **r\_reason\_id** is **AAAAAAAABAAAAAA** and whose **r\_reason\_sk** is greater than or equal to 4.

```
SELECT * FROM tpcds.reason_p WHERE r_reason_id='AAAAAAAABAAAAAA' EXCEPT SELECT * FROM
tpcds.reason_p WHERE r_reason_sk<4;
```

```
r_reason_sk | r_reason_id | r_reason_desc
```

```
-----+-----+-----
 10 | AAAAAAABAAAAAA | reason 2
 10 | AAAAAAABAAAAAA | reason 5
 10 | AAAAAAABAAAAAA | reason 4
```

```
4 | AAAAAAAAAAAAAAAAAA | reason 3
(4 rows)
```

Specify the operator (+) in the **WHERE** clause to indicate a left join.

```
select t1.sr_item_sk,t2.c_customer_id from store_returns t1, customer t2 where t1.sr_customer_sk =
t2.c_customer_sk(+)
order by 1 desc limit 1;
sr_item_sk | c_customer_id
-----+-----
18000 |
(1 row)
```

Specify the operator (+) in the **WHERE** clause to indicate a right join.

```
select t1.sr_item_sk,t2.c_customer_id from store_returns t1, customer t2 where t1.sr_customer_sk(+) =
t2.c_customer_sk
order by 1 desc limit 1;
sr_item_sk | c_customer_id
-----+-----
| AAAAAAAJINGEBAAA
(1 row)
```

Specify the operator (+) in the **WHERE** clause to indicate a left join and add a join condition.

```
select t1.sr_item_sk,t2.c_customer_id from store_returns t1, customer t2 where t1.sr_customer_sk =
t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1 order by 1 limit 1;
sr_item_sk | c_customer_id
-----+-----
1 |
(1 row)
```

If the operator (+) is specified in the **WHERE** clause, do not use expressions connected through **AND/OR**.

```
select t1.sr_item_sk,t2.c_customer_id from store_returns t1, customer t2 where not(t1.sr_customer_sk =
t2.c_customer_sk(+) and t2.c_customer_sk(+) < 1);
ERROR: Operator "+" can not be used in nesting expression.
LINE 1: ...tomer_id from store_returns t1, customer t2 where
not(t1.sr_...
```

If the operator (+) is specified in the **WHERE** clause which does not support expression macros, an error will be reported.

```
select t1.sr_item_sk,t2.c_customer_id from store_returns t1, customer t2 where (t1.sr_customer_sk =
t2.c_customer_sk(+))::bool;
ERROR: Operator "+" can only be used in common expression.
```

If the operator (+) is specified on both sides of an expression in the **WHERE** clause, an error will be reported.

```
select t1.sr_item_sk,t2.c_customer_id from store_returns t1, customer t2 where t1.sr_customer_sk(+) =
t2.c_customer_sk(+);
ERROR: Operator "+" can't be specified on more than one relation in one join condition
HINT: "t1", "t2"...are specified Operator "+" in one condition.
```

## Example: Use GROUP BY for a Query (ROLLUP, CUBE, and GROUPING SETS as Part of GROUP BY)

In some report scenarios, data is often grouped using GROUP BY. In the following example, obtain the number of employees in level-2 departments under a level-1 department.

```
DROP TABLE IF EXISTS emp;
CREATE TABLE emp(
```

```
id int primary key, --Employee ID
name text, --Employee name
dep_1 text, --Level-1 department
dep_2 text --Level-2 department
);
```

```
INSERT INTO emp VALUES
(01, 'liwei', 'Product', 'SRE'),
(02, 'liming', 'Product', 'U1'),
(03, 'hanlei', 'Product', 'U1'),
(04, 'hanmeimei', 'Product', 'SRE'),
(05, 'dingjun', 'Product', 'SRE'),
(06, 'huxue', 'Product', 'E1'),
(07, 'liuyi', 'Product', 'E1'),
(08, 'luhua', 'Product', 'E1'),
(09, 'yangjing', 'Product', 'U1'),
(10, 'binbin', 'Product', 'U1'),
(11, 'jim', 'Product', 'SRE'),
(12, 'leo', 'Product', 'SER');
```

```
SELECT COUNT(*), dep_2 FROM emp GROUP BY dep_2; --Count the number of employees in level-2
departments.
```

| count | dep_2 |
|-------|-------|
| 1     | SER   |
| 3     | E1    |
| 4     | SRE   |
| 4     | U1    |

In common statistical report services, the total number of employees in a level-1 department (that is, the sum of employees in level-2 departments) needs to be calculated. In this case, **ROLLUP** can be used to ensure that there is an additional row of total data in the result, as shown in the following command output. In the following figure, **count** is **12** and **dep\_2** is **Null**, which indicate that the total number of employees in the level-1 department is 12.

```
SELECT COUNT(*), dep_2 FROM emp GROUP BY ROLLUP(dep_2) ORDER BY 1;
```

| count | dep_2  |
|-------|--------|
| 1     | SER    |
| 3     | E1     |
| 4     | U1     |
| 4     | SRE    |
| 12    | (Null) |

The following describes how to use ROLLUP, CUBE, and GROUPING SETS.

- **ROLLUP:**

Create **customer** table.

```
DROP TABLE IF EXISTS customer;
CREATE TABLE customer
(
 c_id char(16) not null, --Customer ID
 c_name char(20), --Customer name
 c_age integer, --Age
 c_country varchar(20), --Country
 c_class char(10), --Customer level
 c_gender text, --Gender
 c_balance numeric); --Balance

INSERT INTO customer VALUES(1, 'tom', '20', 'Region A', '1', 'male', 300);
INSERT INTO customer VALUES(2, 'jack', '30', 'Region B', '1', 'male', 100);
INSERT INTO customer VALUES(3, 'rose', '40', 'Region C', '1', 'female', 200);
INSERT INTO customer VALUES(4, 'Frank', '60', 'Region D', '1', 'male', 100);
INSERT INTO customer VALUES(5, 'Leon', '20', 'Region A', '2', 'male', 200);
INSERT INTO customer VALUES(6, 'Lucy', '20', 'Region A', '1', 'female', 500);
```

**ROLLUP** generates subtotals and grand totals across multiple levels of aggregation within a single query. It starts from the most detailed level of aggregation and progressively rolls up to less detailed levels, ultimately producing a grand total.

```
-- Group by c_country and c_class, then group by c_country, and finally group by the entire table.
SELECT c_country, c_class, sum(c_balance) FROM customer GROUP BY ROLLUP(c_country, c_class)
ORDER BY 1,2,3;
```

| c_country | c_class | sum  |
|-----------|---------|------|
| Region A  | 1       | 800  |
| Region A  | 2       | 200  |
| Region A  | (Null)  | 1000 |
| Region B  | 1       | 100  |
| Region B  | (Null)  | 100  |
| Region C  | 1       | 200  |
| Region C  | (Null)  | 200  |
| Region D  | 1       | 100  |
| Region D  | (Null)  | 100  |
| (Null)    | (Null)  | 1400 |

Total Records: 10

The preceding statement is equivalent to:

```
SELECT c_country, c_class, SUM(c_balance) FROM customer GROUP BY c_country, c_class UNION ALL
SELECT c_country, null, SUM(c_balance) FROM customer GROUP BY c_country
UNION ALL
SELECT null, null, SUM(c_balance) FROM customer ORDER BY 1,2,3;
```

- **CUBE:**

CUBE can be understood as all dimensions, that is, all combinations of columns in the aggregation key.

```
-- Perform GROUP BY on (c_country,c_class), (c_country), and (c_class) in sequence, and on the entire table.
SELECT c_country, c_class, SUM(c_balance) FROM customer GROUP BY CUBE(c_country, c_class)
ORDER BY 1,2,3;
```

| c_country | c_class | sum  |
|-----------|---------|------|
| Region A  | 1       | 800  |
| Region A  | 2       | 200  |
| Region A  | (Null)  | 1000 |
| Region B  | 1       | 100  |
| Region B  | (Null)  | 100  |
| Region C  | 1       | 200  |
| Region C  | (Null)  | 200  |
| Region D  | 1       | 100  |
| Region D  | (Null)  | 100  |
| (Null)    | 1       | 1200 |
| (Null)    | 2       | 200  |
| (Null)    | (Null)  | 1400 |

Total Records: 12

The preceding statement is equivalent to:

```
SELECT c_country, c_class, SUM(c_balance) FROM customer GROUP BY c_country, c_class
union all
SELECT c_country, null, SUM(c_balance) FROM customer GROUP BY c_country
union all
SELECT null, null, SUM(c_balance) FROM customer
union all
SELECT NULL, c_class, SUM(c_balance) FROM customer GROUP BY c_class order by 1,2,3;
```

- **GROUPING SETS:**

**GROUPING SETS** is different from **ROLLUP** and **CUBE**. It extracts some records from the results of **ROLLUP** and **CUBE**.

```
-- Perform GROUP BY on (c_class) and (c_country) separately.
SELECT c_country, c_class, SUM(c_balance) FROM customer GROUP BY GROUPING SETS(c_country,
c_class) ORDER BY 1,2,3;
```

| c_country | c_class | sum  |
|-----------|---------|------|
| Region A  | (Null)  | 1000 |
| Region B  | (Null)  | 100  |
| Region C  | (Null)  | 200  |
| Region D  | (Null)  | 100  |
| (Null)    | 1       | 1200 |
| (Null)    | 2       | 200  |

Total Records: 6

The preceding statement is equivalent to:

```
SELECT c_country, null, SUM(c_balance) FROM customer GROUP BY c_country
UNION ALL
SELECT null, c_class, sum(c_balance) FROM customer GROUP BY c_class ORDER BY 1,2,3;
```

## 21.16.3 SELECT INTO

### Function

**SELECT INTO** defines a new table based on a query result and insert data obtained by query to the new table.

Different from **SELECT**, data found by **SELECT INTO** is not returned to the client. The table columns have the same names and data types as the output columns of the **SELECT**.

### Precautions

**CREATE TABLE AS** provides functions similar to **SELECT INTO** in functions and provides a superset of functions provided by **SELECT INTO**. You are advised to replace **SELECT INTO** with **CREATE TABLE AS** because **SELECT INTO** cannot be used in stored procedures and **SELECT INTO (column)** cannot receive blank rows.

### Syntax

```
[WITH [RECURSIVE] with_query [, ...]]
SELECT [ALL | DISTINCT [ON (expression [, ...])]]
 { * | {expression [[AS] output_name] [, ...] }
 INTO [[GLOBAL | LOCAL | VOLATILE] { TEMPORARY | TEMP } | UNLOGGED] [TABLE] new_table
 [FROM from_item [, ...]]
 [WHERE condition]
 [GROUP BY expression [, ...]]
 [HAVING condition [, ...]]
 [WINDOW {window_name AS (window_definition)} [, ...]]
 [{ UNION | INTERSECT | EXCEPT | MINUS } [ALL | DISTINCT] select]
 [ORDER BY {expression [[ASC | DESC | USING operator] | nlssort_expression_clause } [NULLS { FIRST |
LAST }]} [, ...]]
 [{ [LIMIT { count | ALL }] [OFFSET start [ROW | ROWS]] | { LIMIT start, { count | ALL } }]
```

```
[FETCH { FIRST | NEXT } [count] { ROW | ROWS } ONLY]
[{FOR { UPDATE | SHARE } [OF table_name [, ...]] [NOWAIT]} [...]];
```

## Parameter Description

**INTO** [ [ **GLOBAL** | **LOCAL** | **VOLATILE** ] { **TEMPORARY** | **TEMP** } | **UNLOGGED** ]  
[ **TABLE** ] **new\_table**

[ **GLOBAL** | **LOCAL** | **VOLATILE** ] { **TEMPORARY** | **TEMP** } specifies the type of a temporary table. For details, see the description of **GLOBAL** | **LOCAL** | **VOLATILE** in [Parameter Description](#).

**UNLOGGED** indicates that the table is created as an unlogged table. Data written to unlogged tables is not written to the write-ahead log, which makes them considerably faster than ordinary tables. However, they are not crash-safe: an unlogged table is automatically truncated after a crash or unclean shutdown. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are automatically unlogged as well.

**new\_table** specifies the name of the new table.

### NOTE

For details about other **SELECT INTO** parameters, see [SELECT Main Clause Parameters](#) in **SELECT**.

## Examples

Add values whose **TABLE\_SK** is less than 3 in the **reason\_t** table to the new table.

```
CREATE TABLE reason_t
(
 TABLE_SK INTEGER ,
 TABLE_ID VARCHAR(20) ,
 TABLE_NA VARCHAR(20)
);
INSERT INTO reason_t VALUES (1, 'S01', 'StudentA'), (2, 'T01', 'TeacherA'), (3, 'T02', 'TeacherB'), (3, 'S02', 'StudentB');

SELECT * INTO reason_t_bck FROM reason_t WHERE TABLE_SK < 3;
INSERT 0 2
```

## Helpful Links

[SELECT](#)

# 21.17 TCL Syntax

## 21.17.1 TCL Syntax Overview

Transaction Control Language (TCL) controls the time and effect of database transactions and monitors the database.

### Commit

DWS uses the **COMMIT** or **END** statement to commit transactions. For details, see [COMMIT](#) | [END](#).



## Setting a Savepoint

DWS creates a new savepoint in the current transaction. For details, see [SAVEPOINT](#).

## Rollback

DWS rolls back the current transaction to the last committed state. For details, see [ROLLBACK](#).

## 21.17.2 ABORT

### Function

**ABORT** rolls back the current transaction and cancels the changes in the transaction.

This command is equivalent to [ROLLBACK](#), and is present only for historical reasons. Now **ROLLBACK** is recommended.

### Precautions

**ABORT** has no impact outside a transaction, but will provoke a warning.

### Syntax

```
ABORT [WORK | TRANSACTION] ;
```

### Parameter Description

**WORK | TRANSACTION**

This keyword is optional and does not affect the **ABORT** operation.

### Examples

Abort a transaction. Performed update operations will be undone.

```
ABORT;
```

### Helpful Links

[SET TRANSACTION](#), [COMMIT | END](#), [ROLLBACK](#)

## 21.17.3 BEGIN

### Function

**BEGIN** may be used to initiate an anonymous block or a single transaction. This section describes the syntax of **BEGIN** used to initiate an anonymous block. For details about the **BEGIN** syntax that initiates transactions, see [START TRANSACTION](#).

An anonymous block is a structure that can dynamically create and execute stored procedure code instead of permanently storing code as a database object in the database.

## Precautions

None

## Syntax

- Enable an anonymous block:  

```
[DECLARE [declare_statements]]
BEGIN
execution_statements
END;
/
```
- -- Start a transaction:  

```
BEGIN [WORK | TRANSACTION]
[
 {
 ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE
 READ }
 | { READ WRITE | READ ONLY }
 } [, ...]
];
```

## Parameter Description

Table 21-309 BEGIN parameters

| Parameter            | Description                                                                           | Value Range                    |
|----------------------|---------------------------------------------------------------------------------------|--------------------------------|
| declare_statements   | Declares a variable, including its name and type, for example, <b>sales_cnt int</b> . | -                              |
| execution_statements | Specifies the statement to be executed in an anonymous block.                         | Use an existing function name. |

## Examples

- Start a transaction block.  

```
BEGIN;
```
- Start a transaction block at the REPEATABLE READ isolation level.  

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```
- Generate a string using an anonymous block.  

```
BEGIN
dbms_output.put_line('Hello');
END;
/
```

## Helpful Links

[START TRANSACTION](#)

## 21.17.4 CHECKPOINT

### Function

A checkpoint is a point in the transaction log sequence at which all data files have been updated to reflect the information in the log. All data files will be flushed to a disk.

**CHECKPOINT** forces a transaction log checkpoint. By default, WALs periodically specify checkpoints in a transaction log. You may use **gs\_guc** to specify run-time parameters **checkpoint\_segments** and **checkpoint\_timeout** to adjust the atomized checkpoint intervals.

### Precautions

- Only a system administrator has the permission to call **CHECKPOINT**.
- CHECKPOINT** forces an immediate checkpoint when the related command is issued, without waiting for a regular checkpoint scheduled by the system.

### Syntax

```
CHECKPOINT;
```

### Parameter Description

None

### Examples

Set a checkpoint:

```
CHECKPOINT;
```

## 21.17.5 COMMIT | END

### Function

**COMMIT** or **END** commits all operations of a transaction.

### Precautions

Only the transaction creators or system administrators can run the **COMMIT** command. The creation and commit operations must be in different sessions.

### Syntax

```
{ COMMIT | END } [WORK | TRANSACTION] ;
```

## Parameter Description

**Table 21-310** COMMIT | END parameters

| Parameter          | Description                                                                                            |
|--------------------|--------------------------------------------------------------------------------------------------------|
| COMMIT   END       | Commits the current transaction and make all changes made by the transaction become visible to others. |
| WORK   TRANSACTION | The keyword is optional and only improves readability, with no effect on transaction commitment.       |

## Examples

Commit the transaction to make all changes permanent:

```
COMMIT;
```

## Helpful Links

[ROLLBACK](#)

## 21.17.6 COMMIT PREPARED

### Function

**COMMIT PREPARED** commits a prepared two-phase transaction.

### Precautions

- The function is only available in maintenance mode (when GUC parameter **xc\_maintenance\_mode** is **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the transaction creators or system administrators can run the **COMMIT** command. The creation and commit operations must be in different sessions.
- The transaction function is maintained automatically by the database, and should be not visible to users.

### Syntax

```
COMMIT PREPARED transaction_id ;
COMMIT PREPARED transaction_id WITH CSN;
```

## Parameter Description

**Table 21-311** COMMIT PREPARED parameters

| Parameter                   | Description                                                                                                                                  |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| transaction_id              | Identifier of the transaction to be committed. The identifier cannot be the same as any identifier used by the current prepared transaction. |
| CSN(commit sequence number) | Sequence number of the transaction to be committed. The value is a 64-bit ascending unsigned number.                                         |

## Helpful Links

[PREPARE TRANSACTION](#), [ROLLBACK PREPARED](#)

## 21.17.7 PREPARE TRANSACTION

### Function

**PREPARE TRANSACTION** prepares the current transaction for two-phase commit.

After this command, the transaction is no longer associated with the current session; instead, its state is fully stored on disk, and there is a high probability that it can be committed successfully, even if a database crash occurs before the commit is requested.

Once prepared, a transaction can later be committed or rolled back with **COMMIT PREPARED** or **ROLLBACK PREPARED**, respectively. Those commands can be issued from any session, not only the one that executed the original transaction.

From the point of view of the issuing session, **PREPARE TRANSACTION** is not unlike a **ROLLBACK** command: after executing it, there is no active current transaction, and the effects of the prepared transaction are no longer visible. (The effects will become visible again if the transaction is committed.)

If the **PREPARE TRANSACTION** command fails for any reason, it becomes a **ROLLBACK** and the current transaction is canceled.

### Precautions

- The transaction function is maintained automatically by the database, and should be not visible to users.
- When running the **PREPARE TRANSACTION** command, increasing the value of **max\_prepared\_transactions** in configuration file **postgresql.conf**. You are advised to set **max\_prepared\_transactions** to a value not less than that of **max\_connections** so that one pending prepared transaction is available for each session.

## Syntax

```
PREPARE TRANSACTION transaction_id;
```

## Parameter Description

### transaction\_id

An arbitrary identifier that later identifies this transaction for **COMMIT PREPARED** or **ROLLBACK PREPARED**. The identifier must be different from those for current prepared transactions.

Value range: The identifier must be written as a string literal, and must be less than 200 bytes long.

## Helpful Links

[COMMIT PREPARED](#), [ROLLBACK PREPARED](#)

## 21.17.8 SAVEPOINT

### Function

**SAVEPOINT** establishes a new savepoint within the current transaction.

A savepoint is a special mark inside a transaction that rolls back all commands that are executed after the savepoint was established, restoring the transaction state to what it was at the time of the savepoint.

### Precautions

- Use **ROLLBACK TO SAVEPOINT** to roll back to a savepoint. Use **RELEASE SAVEPOINT** to destroy a savepoint but keep the effects of the commands executed after the savepoint was established.
- Savepoints can only be established when inside a transaction block. There can be multiple savepoints defined within a transaction.
- **SAVEPOINT** cannot be used for functions, anonymous blocks, or stored procedures.
- In the case of an unexpected termination of a distributed thread or process caused by a node or connection failure, or of an error caused by the inconsistency between source and destination table structures in a COPY FROM operation, the transaction cannot be rolled back to the established savepoint. Instead, the entire transaction will be rolled back.
- According to the SQL standard, a savepoint is destroyed automatically when another savepoint with the same name is established. In DWS, old savepoints are kept, though only the most recent one will be used for rollback or release. Releasing the newer savepoint with **RELEASE SAVEPOINT** will cause the older one to again become accessible to **ROLLBACK TO SAVEPOINT** and **RELEASE SAVEPOINT**. Except for this, **SAVEPOINT** is fully SQL conforming.

## Syntax

```
SAVEPOINT savepoint_name;
```

## Parameter Description

### **savepoint\_name**

Specifies the name of a new savepoint.

## Examples

- Create a savepoint and undo all commands executed after the savepoint is created:

```
START TRANSACTION;
INSERT INTO table1 VALUES (1);
SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (2);
ROLLBACK TO SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (3);
COMMIT;
```

Query the table content, which should contain 1 and 3 but not 2, because 2 has been rolled back.

- Create and then destroy a savepoint.

```
START TRANSACTION;
INSERT INTO table1 VALUES (3);
SAVEPOINT my_savepoint;
INSERT INTO table1 VALUES (4);
RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

Query the table content, which should contain both 3 and 4.

## Helpful Links

[RELEASE SAVEPOINT, ROLLBACK TO SAVEPOINT](#)

## 21.17.9 SET TRANSACTION

### Function

**SET TRANSACTION** sets the characteristics of the current transaction. It has no effect on any subsequent transactions. Available transaction characteristics include the transaction separation level and transaction access mode (read/write or read only).

### Precautions

None

### Syntax

Set the isolation level and access mode of the transaction.

```
{ SET [LOCAL] TRANSACTION|SET SESSION CHARACTERISTICS AS TRANSACTION }
{ ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
| { READ WRITE | READ ONLY } } [, ...]
```

## Parameter Description

**Table 21-312** SET TRANSACTION parameters

| Parameter              | Description                                                                                                                                                                                                                                                                                                                                                                  | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOCAL                  | Indicates that the specified command takes effect only for the current transaction.                                                                                                                                                                                                                                                                                          | -                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| SESSION                | Indicates that the specified command takes effect only for the current session.                                                                                                                                                                                                                                                                                              | A string, which must comply with the naming convention.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| ISOLATION LEVEL        | <p>Specifies the transaction isolation level that determines the data that a transaction can view if other concurrent transactions exist.</p> <p><b>NOTE</b><br/>The isolation level of a transaction cannot be reset after the first clause (<b>INSERT</b>, <b>DELETE</b>, <b>UPDATE</b>, <b>FETCH</b>, <b>COPY</b>) for modifying data is executed in the transaction.</p> | <ul style="list-style-type: none"><li>• <b>READ COMMITTED</b>: Only committed data is read. The default value is <b>READ COMMITTED</b>.</li><li>• <b>READ UNCOMMITTED</b>: DWS does not support <b>READ UNCOMMITTED</b>. If <b>READ UNCOMMITTED</b> is set, <b>READ COMMITTED</b> is executed instead.</li><li>• <b>REPEATABLE READ</b>: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.</li><li>• <b>SERIALIZABLE</b>: DWS does not support <b>SERIALIZABLE</b>. If <b>SERIALIZABLE</b> is set, <b>REPEATABLE READ</b> is executed instead.</li></ul> |
| READ WRITE   READ ONLY | Specifies the transaction access mode.                                                                                                                                                                                                                                                                                                                                       | Read-write mode or read-only mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## Examples

Set the isolation level of the current transaction to **READ COMMITTED** and the access mode to **READ ONLY**:

```
START TRANSACTION;
SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
COMMIT;
```



# 21.17.10 START TRANSACTION

## Function

**START TRANSACTION** starts a transaction. If the isolation level, read/write mode, or deferrable mode is specified, a new transaction will have those characteristics. You can also specify them using [SET TRANSACTION](#).

## Precautions

None

## Syntax

Format 1: START TRANSACTION

```
START TRANSACTION
[
{
ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
| { READ WRITE | READ ONLY }
} [, ...]
];
```

Format 2: BEGIN

```
BEGIN [WORK | TRANSACTION]
[
{
ISOLATION LEVEL { READ COMMITTED | READ UNCOMMITTED | SERIALIZABLE | REPEATABLE READ }
| { READ WRITE | READ ONLY }
} [, ...]
];
```

## Parameter Description

Table 21-313 START TRANSACTION parameters

| Parameter          | Description                                                                       | Value Range |
|--------------------|-----------------------------------------------------------------------------------|-------------|
| WORK   TRANSACTION | Optional keyword in the <b>BEGIN</b> format, which does not affect the operation. | -           |

| Parameter                 | Description                                                                                                                                                                                                                                                                                                                                                                  | Value Range                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ISOLATION LEVEL           | <p>Transaction isolation level. This parameter determines the data that a transaction can view if other concurrent transactions exist.</p> <p><b>NOTE</b></p> <p>The isolation level of a transaction cannot be reset after the first clause (<b>INSERT</b>, <b>DELETE</b>, <b>UPDATE</b>, <b>FETCH</b>, <b>COPY</b>) for modifying data is executed in the transaction.</p> | <ul style="list-style-type: none"><li>• <b>READ COMMITTED</b>: Only committed data is read. The default value is <b>READ COMMITTED</b>.</li><li>• <b>READ UNCOMMITTED</b>: DWS does not support <b>READ UNCOMMITTED</b>. If <b>READ UNCOMMITTED</b> is set, <b>READ COMMITTED</b> is executed instead.</li><li>• <b>REPEATABLE READ</b>: Only the data committed before transaction start is read. Uncommitted data or data committed in other concurrent transactions cannot be read.</li><li>• <b>SERIALIZABLE</b>: DWS does not support <b>SERIALIZABLE</b>. If <b>SERIALIZABLE</b> is set, <b>REPEATABLE READ</b> is executed instead.</li></ul> |
| READ WRITE<br>  READ ONLY | Transaction access mode.                                                                                                                                                                                                                                                                                                                                                     | Read-write mode or read-only mode.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## Examples

- Start a transaction in default mode.  

```
START TRANSACTION;
SELECT * FROM tpceds.reason;
END;
```
- Start a transaction with the isolation level being **READ COMMITTED** and the access mode being **READ WRITE**:  

```
START TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SELECT * FROM tpceds.reason;
COMMIT;
```

## Helpful Links

[COMMIT | END, ROLLBACK, SET TRANSACTION](#)

## 21.17.11 ROLLBACK

### Function

Rolls back the current transaction and backs out all updates in the transaction.

**ROLLBACK** backs out of all changes that a transaction makes to a database if the transaction fails to be executed due to a fault.

### Precautions

If a **ROLLBACK** statement is executed out of a transaction, no error occurs, but a warning information is displayed.

## Syntax

```
ROLLBACK [WORK | TRANSACTION];
```

## Parameter Description

### WORK | TRANSACTION

This keyword is optional and only improves readability, with no effect on the **ROLLBACK** operation.

## Examples

Undo all changes in the current transaction:

```
ROLLBACK;
```

## Helpful Links

[COMMIT | END](#)

## 21.17.12 RELEASE SAVEPOINT

### Function

**RELEASE SAVEPOINT** destroys a savepoint previously defined in the current transaction.

Destroying a savepoint makes it unavailable as a rollback point, but it has no other user visible behavior. It does not undo the effects of commands executed after the savepoint was established. To do that, use **ROLLBACK TO SAVEPOINT**. Destroying a savepoint when it is no longer needed allows the system to reclaim some resources earlier than transaction end.

**RELEASE SAVEPOINT** also destroys all savepoints that were established after the named savepoint was established.

### Precautions

- Specifying a savepoint name that was not previously defined causes an error.
- It is not possible to release a savepoint when the transaction is in an aborted state.
- If multiple savepoints have the same name, only the one that was most recently defined is released.

## Syntax

```
RELEASE [SAVEPOINT] savepoint_name;
```

## Parameter Description

### *savepoint\_name*

Specifies the name of the savepoint you want to destroy.

## Examples

Create and then destroy a savepoint:

```
BEGIN;
 INSERT INTO tpcds.table1 VALUES (3);
 SAVEPOINT my_savepoint;
 INSERT INTO tpcds.table1 VALUES (4);
 RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

## Helpful Links

[SAVEPOINT](#), [ROLLBACK TO SAVEPOINT](#)

## 21.17.13 ROLLBACK PREPARED

### Function

**ROLLBACK PREPARED** cancels a transaction ready for two-phase committing.

### Precautions

- The function is only available in maintenance mode (when GUC parameter **xc\_maintenance\_mode** is **on**). Exercise caution when enabling the mode. It is used by maintenance engineers for troubleshooting. Common users should not use the mode.
- Only the user that initiates a transaction or the system administrator can roll back the transaction.
- The transaction function is maintained automatically by the database, and should be not visible to users.

### Syntax

```
ROLLBACK PREPARED transaction_id ;
```

### Parameter Description

**transaction\_id**

Specifies the identifier of the transaction to be submitted. The identifier must be different from those for current prepared transactions.

## Helpful Links

[COMMIT PREPARED](#), [PREPARE TRANSACTION](#)

## 21.17.14 ROLLBACK TO SAVEPOINT

### Function

**ROLLBACK TO SAVEPOINT** rolls back to a savepoint. It implicitly destroys all savepoints that were established after the named savepoint.

Rolls back all commands that were executed after the savepoint was established. The savepoint remains valid and can be rolled back to again later, if needed.

## Precautions

- Specifying a savepoint name that has not been established is an error.
- Cursors have somewhat non-transactional behavior with respect to savepoints. Any cursor that is opened inside a savepoint will be closed when the savepoint is rolled back. If a previously opened cursor is affected by a **FETCH** or **MOVE** command inside a savepoint that is later rolled back, the cursor remains at the position that **FETCH** left it pointing to (that is, the cursor motion caused by **FETCH** is not rolled back). Closing a cursor is not undone by rolling back, either. A cursor whose execution causes a transaction to abort is put in a cannot-execute state, so while the transaction can be restored using **ROLLBACK TO SAVEPOINT**, the cursor can no longer be used.
- Use **ROLLBACK TO SAVEPOINT** to roll back to a savepoint. Use **RELEASE SAVEPOINT** to destroy a savepoint but keep the effects of the commands executed after the savepoint was established.

## Syntax

```
ROLLBACK [WORK | TRANSACTION] TO [SAVEPOINT] savepoint_name;
```

## Parameter Description

### **savepoint\_name**

Rolls back the name of the specified savepoint.

## Examples

Undo the effects of the commands executed after my\_savepoint was established.

```
ROLLBACK TO SAVEPOINT my_savepoint;
```

Cursor positions are not affected by savepoint rollback:

```
BEGIN;
DECLARE foo CURSOR FOR SELECT 1 UNION SELECT 2;
SAVEPOINT foo;
FETCH 1 FROM foo;
?column?

1
ROLLBACK TO SAVEPOINT foo;
FETCH 1 FROM foo;
?column?

2
COMMIT;
```

## Helpful Links

[SAVEPOINT](#), [RELEASE SAVEPOINT](#)