

# Data Warehouse Service

## FAQs

**Issue** 03  
**Date** 2023-11-21



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2023. 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

---

<b>1 General Problems.....</b>	<b>1</b>
1.1 Why Are Data Warehouses Necessary?.....	1
1.2 What Are the Differences Between a Data Warehouse and the Hadoop Big Data Platform?.....	2
1.3 Why Should I Use Public Cloud GaussDB(DWS)?.....	3
1.4 Should I Choose Public Cloud GaussDB(DWS) or RDS?.....	3
1.5 When Should I Use GaussDB(DWS) and MRS?.....	4
1.6 What Is the User Quota?.....	4
1.7 What Are the Differences Between Users and Roles?.....	4
1.8 How Do I Check the Creation Time of a Database User?.....	6
1.9 Regions and AZs.....	7
1.10 Is My Data Secure in GaussDB(DWS)?.....	8
1.11 How Is GaussDB(DWS) Secured?.....	9
1.12 Can I Modify the Security Group of a GaussDB(DWS) Cluster?.....	9
1.13 How Are LibrA, GaussDB A, and GaussDB(DWS) Related?.....	10
1.14 What Is a Database/Data Warehouse/Data Lake/Lakehouse?.....	10
<b>2 Cluster Management.....</b>	<b>14</b>
2.1 What Do I Do If Creating a GaussDB(DWS) Cluster Failed?.....	14
2.2 How Can I Clear and Reclaim the Storage Space?.....	14
2.3 Can I Switch My Cluster Nodes to Another Region After Purchase?.....	15
2.4 Why Did the Used Storage Shrink After Scale-out?.....	15
2.5 How Do I View Node Metrics (CPU, Memory, and Disk Usage)?.....	16
2.6 Does GaussDB(DWS) Support BMS?.....	16
2.7 How Is the Disk Space or Capacity of GaussDB(DWS) Calculated?.....	16
2.8 What Are the gaussdb and postgres Databases of GaussDB(DWS)?.....	17
2.9 How Do I Set the Maximum Number of Sessions When Adding an Alarm Rule on Cloud Eye?.....	17
2.10 What Should I Do If the Scale-out Check Fails?.....	18
2.11 When Should I Add CNs or Scale out a cluster?.....	19
2.12 What Are the Scenarios of Resizing a Cluster, Changing the Node Flavor, Scale-out, and Scale-in?...	19
2.13 How Should I Select from a Small-Flavor Many-Node Cluster and a Large-Flavor Three-Node Cluster with Same CPU Cores and Memory?.....	21
2.14 What Are the Differences Between Cloud SSDs and Local SSDs?.....	21
2.15 What Are the Differences Between Hot Data Storage and Cold Data Storage?.....	21
<b>3 Database Connections.....</b>	<b>23</b>

3.1 How Applications Communicate with GaussDB(DWS)?.....	23
3.2 Does GaussDB(DWS) Support Third-Party Clients and JDBC and ODBC Drivers?.....	28
3.3 Can I Connect to GaussDB(DWS) Cluster Nodes Using SSH?.....	28
3.4 What Should I Do If I Cannot Connect to a Data Warehouse Cluster?.....	28
3.5 Why Was I Not Notified of Failure Unbinding the EIP When GaussDB(DWS) Is Connected Over the Internet?.....	29
3.6 How Do I Configure a Whitelist to Protect Clusters Available Through a Public IP Address?.....	30
<b>4 Data Import and Export.....</b>	<b>31</b>
4.1 What Are the Differences Between Data Formats Supported by OBS and GDS Foreign Tables?.....	31
4.2 How Do I Import Incremental Data Using an OBS Foreign Table?.....	31
4.3 How Can I Import Data to GaussDB(DWS)?.....	31
4.4 How Much Service Data Can a Data Warehouse Store?.....	32
4.5 How Do I Use \Copy to Import and Export Data?.....	32
4.6 How Do I Implement Fault Tolerance Import Between Different Encoding Libraries.....	33
4.7 Can I Import and Export Data to and from OBS Across Regions?.....	34
4.8 How Do I Import GaussDB(DWS)/Oracle/MySQL/SQL Server Data to GaussDB(DWS) (Whole Database Migration)?.....	34
4.9 Can I Import Data over the Public/External Network Using GDS?.....	35
4.10 Which Are the Factors That Affect GaussDB(DWS) Import Performance?.....	35
<b>5 Account, Password, and Permissions.....</b>	<b>36</b>
5.1 How Does GaussDB(DWS) Implement Workload Isolation?.....	36
5.2 How Do I Change the Password of a Database Account When the Password Expires?.....	40
5.3 How Do I Grant Table Permissions to a User?.....	41
5.4 How Do I Grant Schema Permissions to a User?.....	45
5.5 How Do I Create a Database Read-only User?.....	47
5.6 How Do I Create Private Database Users and Tables?.....	48
5.7 How Do I Revoke the CONNECT ON DATABASE Permission from a User?.....	49
5.8 How Do I View the Table Permissions of a User?.....	50
5.9 Who Is User Ruby?.....	53
<b>6 Database Usage.....</b>	<b>54</b>
6.1 How Do I Change Distribution Columns?.....	54
6.2 How Do I View and Set the Database Character Encoding?.....	56
6.3 What Do I Do If Date Type Is Automatically Converted to the Timestamp Type During Table Creation?.....	57
6.4 Do I Need to Run VACUUM FULL and ANALYZE on Common Tables Periodically?.....	58
6.5 Do I Need to Set a Distribution Key After Setting a Primary Key?.....	60
6.6 Is GaussDB(DWS) Compatible with PostgreSQL Stored Procedures?.....	60
6.7 What Are Partitioned Tables, Partitions, and Partition Keys?.....	61
6.8 How Can I Export the Table Structure?.....	61
6.9 How Can I Delete Table Data Efficiently?.....	61
6.10 How Do I View Foreign Table Information?.....	63
6.11 If No Distribution Column Is Specified, How Will Data Be Stored?.....	63

6.12 How Do I Replace the Null Result with 0?.....	64
6.13 How Do I Check Whether a Table Is Row-Stored or Column-Stored?.....	65
6.14 How Do I Query the Information About GaussDB(DWS) Column-Store Tables?.....	66
6.15 Why Sometimes the GaussDB(DWS) Query Indexes Become Invalid?.....	67
6.16 How Do I Use a User-Defined Function to Rewrite the CRC32() Function?.....	75
6.17 What Are the Schemas Starting with <b>pg_toast_temp*</b> or <b>pg_temp*</b> ?.....	76
6.18 Solutions to Inconsistent GaussDB(DWS) Query Results.....	76
6.19 Which System Catalogs That the VACUUM FULL Operation Cannot Be Performed on?.....	81
6.20 In Which Scenarios Would a Statement Be "idle in transaction"?.....	83
6.21 How Does GaussDB(DWS) Implement Row-to-Column and Column-to-Row Conversion?.....	85
6.22 What Are the Differences Between Unique Constraints and Unique Indexes?.....	88
<b>7 Database Performance.....</b>	<b>90</b>
7.1 Why Is SQL Execution Slow After Long GaussDB(DWS) Usage?.....	90
7.2 Why Does GaussDB(DWS) Perform Worse Than a Single-Server Database in Extreme Scenarios?.....	91
7.3 How Can I View SQL Execution Records in a Certain Period When Read and Write Requests Are Blocked?.....	91
7.4 What Do I Do If My Cluster Is Unavailable Because of Insufficient Space?.....	92
7.5 What is Operator Spilling in GaussDB(DWS)?.....	92
7.6 GaussDB(DWS) CPU Resource Management.....	93
7.7 Why the Tasks Executed by an Ordinary User Are Slower Than That Executed by the dbadmin User?.....	96
7.8 What Are the Factors Related to the Single-Table Query Performance in GaussDB(DWS)?.....	97
<b>8 Snapshot Backup and Restoration.....</b>	<b>99</b>
8.1 Why Is Creating an Automated Snapshot So Slow?.....	99
8.2 Does a DWS Snapshot Have the Same Function as an EVS Snapshot?.....	99
<b>9 Billing.....</b>	<b>101</b>
9.1 How Do I Renew the Service?.....	101
9.2 Is Refund Supported?.....	102
9.3 How Am I Billed for Scheduled Synchronization of GaussDB(DWS) Data to a PostgreSQL Database?.....	102
9.4 How Can I Try Out GaussDB(DWS) for Free?.....	102
9.5 Why Was I Deducted Fees After My GaussDB(DWS) Free Trial Expired?.....	103
9.6 Why Can't I See a Cluster After I Subscribe to a Free GaussDB(DWS) Trial?.....	103
9.7 How Can I Stop GaussDB(DWS) Billing?.....	103
9.8 Does Pay-per-Use Billing Stop When My Cluster Stops?.....	104
9.9 Why Is the Purchase Button Unavailable When I Create a Cluster?.....	104
9.10 How Do I Unfreeze a Cluster?.....	105
9.11 Can I Freeze or Shut Down a GaussDB(DWS) Cluster to Stop Billing?.....	105

# 1 General Problems

---

## 1.1 Why Are Data Warehouses Necessary?

### Status Quo and Requirements

Much data (orders, stocks, materials, and payments) is generated in the business operation systems and background (transactional) database of enterprises.

Decision makers categorize and analyze the data for business decision-making.

### Difficulties

Data categorization and analysis involve the concurrent access to the data in multiple database tables. That is, multiple tables being updated by different transactions may be locked at the same time, which may cause complications to the database systems during peak hours.

- Locking multiple tables increases the latency of complex query.
- The transactions that are updating the database tables are blocked, causing delays or interruptions.

### Solution

Data warehouses excel in data aggregation and association, so users mine more data, get more information, and make better decisions. The mining requires complex queries that involve data on multiple tables.

The ETL process copies data in business operation databases to data warehouses for analysis and computing. Data can be aggregated from multiple business operation systems into one data warehouse for better association, analysis, and actionable insights.

Data warehouses and standard transaction-oriented databases such as Oracle, Microsoft SQL Server, and MySQL use different design modes. Data warehouses are optimized in terms of data aggregation and association but the transaction or data adding and deleting functions or performance may not be guaranteed. Therefore, data warehouses and databases apply to different scenarios.

Transactional databases are dedicated to transaction processing (business operation of enterprises) whereas data warehouses excel at complex data analysis. In conclusion, databases apply to data updates whereas data warehouses apply to data analysis.

## 1.2 What Are the Differences Between a Data Warehouse and the Hadoop Big Data Platform?

The Hadoop big data platform can be regarded as a next-generation data warehousing system. It has the characteristics of modern data warehouses and is widely used by enterprises. Because of the scalability of MPP, the MPP-based data warehousing system is sometimes classified as a big data platform.

However, data warehouses greatly differ from the Hadoop platform in function and user experience in different scenarios. For details, see the following table.

**Table 1-1** Feature comparison between data warehouses and the Hadoop big data platform

Feature	Hadoop	Data Warehouse
Number of compute nodes	1000s	Max 256
Data volume	Over 10 PB	Max 10 PB
Data type	Relational, semi-relational, unstructured (voice, images, and video)	Relational only
Latency	Medium to high	Low
Application ecosystem	Innovative/AI	Traditional/BI
Application development API	SQL and other programming language APIs, such as MapReduce	Standard database SQL
Scalability	Unlimited, with comprehensive programming APIs	Limited, supported by UDFs
Transaction support	Limited	Comprehensive

Data warehouses and the Hadoop platform work together in different scenarios. GaussDB(DWS) on the public cloud can seamlessly integrate with Hadoop-based MRS on the public cloud to provide the SQL-over-Hadoop data sharing across platforms and services. GaussDB(DWS) serves as a data warehouse for managing massive data while relishing the openness, convenience, and innovation of the Hadoop platform. You can also enjoy the upper-layer applications of conventional data warehouses, especially BI applications, using GaussDB(DWS).

## 1.3 Why Should I Use Public Cloud GaussDB(DWS)?

Conventional data warehouses are not practical for smaller enterprises due to high cost, time-consuming device and system selection and procurement, and complex scale-out.

GaussDB(DWS) on the public cloud is the better choice:

- This cloud service of distributed MPP data warehousing is very open, efficient, compatible, scalable, and is easy to O&M.
- Developed on the FusionInsight LibrA data warehouse kernel, it empowers public cloud enterprises with better and consistent experience on and off the cloud.

FusionInsight LibrA is a next-generation distributed data warehousing system with independent intellectual property rights. Currently, it is widely used in government, finance, and carriers. FusionInsight LibrA is compatible with mainstream open-source Postgres databases, especially in Oracle and Teradata SQL statements. GaussDB(DWS) engineers have designed a kernel of hybrid row-column stores not only for faster analysis but also for data processing, such as adding, deleting, modifying data. FusionInsight LibrA features the cost optimizer and warehouse technologies, including machine code vector computing and inter/intra-parallelism for operators and nodes. It uses LLVM to optimize the local code in compilation query plans. More powerful data query and analysis addresses service pain points and improves user experience.

- GaussDB(DWS) can be used out of the box.

Applying for GaussDB(DWS) on the public cloud takes only a few minutes, freeing you from the time consuming process of searching for and purchasing data warehouses. This not only simplifies the procurement, but also lowers the cost and requirements for using data warehouses. Small and medium-sized enterprises with access to GaussDB(DWS) can seamlessly mine data values for their development and form actionable insights.

## 1.4 Should I Choose Public Cloud GaussDB(DWS) or RDS?

Both allow you to run conventional relational databases on the cloud and transfer database management loads. RDS databases are useful for OLTP, reporting, and analysis, but are less capable of handling read operations of a large amount of data (complex read-only queries). GaussDB(DWS) is useful for OLAP by reducing analysis and report workloads of large data sets by an order of magnitude, thanks to its multi-node scale and resources and optimized algorithms (**column storage, vectorized executors, and distributed frameworks**).

You can scale out a GaussDB(DWS) cluster to address complex data and queries, or to handle overwhelming analysis and report workloads that affect OLTP performance.

The following table shows the comparison between OLTP and OLAP.



**Table 1-2** Feature comparison between OLTP and OLAP

Feature	RDS for OLTP	GaussDB(DWS) for OLAP
User	Operations and low-level management	Decision-makers and senior management
Function	Daily operation processing	Analysis and decision-making
Design	By application	By theme
Data	Latest, detailed, two-dimensional, discrete	Historical, integrated, multidimensional, unified
Access	Dozens of read and write records	Millions of read records
Coverage	Simple read/write operations	Complex queries
Database size	Hundreds of GB	TB or PB

## 1.5 When Should I Use GaussDB(DWS) and MRS?

MRS works better with big data processing frameworks such as Apache Spark, Hadoop, and HBase, to process and analyze ultra-large data sets through custom code. It allows you to control cluster configurations and software installed in the cluster.

GaussDB(DWS) works better with complex queries of a large amount of structured data. It aims to pool data from different sources together, such as inventory, finance, and retail system. To ensure consistency and accuracy of enterprise reports, GaussDB(DWS) stores data in a highly structured manner. This structure can directly build the data consistency rule to the database table. Additionally, GaussDB(DWS) is highly compatible with standard SQL statements and the syntax of conventional transaction-supported databases.

GaussDB(DWS) is preferred when you want to perform complex query of a large amount of structured data with high performance.

## 1.6 What Is the User Quota?

For services, quotas limit the number of resources available to users. If you need more, submit a service ticket to increase your quotas. Once approved, we will update your resource quota accordingly and send you a notification. For details about quota operations, see [Quotas](#).

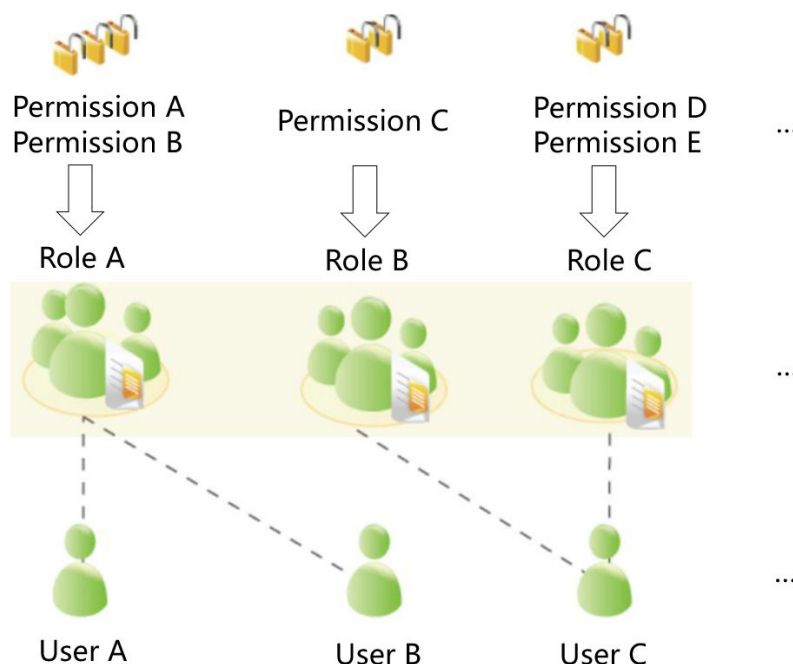
## 1.7 What Are the Differences Between Users and Roles?

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.

- A role is a set of permissions. Generally, roles are used to sort permissions. Users are used to manage permissions and perform operations.
- A role can inherit permissions from other roles. All users in a user group automatically inherit the operation permissions of the role of the group.
- In a database, the permissions of users come from roles.
- A user group is a group of users who have the same permission.
- A user can be regarded as a role with the login permission.
- A role can be regarded as a user without the login permission.

The permissions provided by Gauss(DWS) include the O&M permissions for components on the management plane. You can assign different permissions to users as needed. The management plane uses roles for better permissions management. You can select specified permissions and assign them to roles in a unified manner. In this way, permissions can be viewed and managed in a centralized manner.

The following figure shows the relationships between permissions, roles, and users in unified permissions management.



GaussDB(DWS) provides various permissions. Select and assign permissions to different users based on service scenarios. A role can be assigned one or more permissions.

After a role is granted to a user through **GRANT**, the user will have all the permissions of the role. It is recommended that roles be used to efficiently grant permissions. A user has permissions only for their own tables, but does not have permissions for other users' tables in their schemas.

- Role A is assigned operation permissions A and B. After role A is allocated to user A and user B, user A and user B can obtain operation permissions A and B.

- Role B is assigned operation permission C. After role B is allocated to user C, user C can obtain operation permissions C.
- Role C is assigned operation permissions D and E. After role C is allocated to user C, user C can obtain operation permissions D and F.

## 1.8 How Do I Check the Creation Time of a Database User?

### Method 1:

When you create a GaussDB(DWS) database user, if the time when the user takes effect (**VALID BEGIN**) is the same as the creation time of the user, and the time when the user takes effect has not been changed, you can check the **valbegin** column in the **PG\_USER** view to check the user creation time.

The following is an example:

Create user **jerry** and set its validity start time to its current creation time.

```
CREATE USER jerry PASSWORD 'password' VALID BEGIN '2022-05-19 10:31:56';
```

View users in the **PG\_USER** view. The **valbegin** column indicates the time when **jerry** took effect, that is, the time when jerry was created.

```
SELECT * FROM PG_USER;
```

username	usesysid	usecreatedb	usesuper	usecatupd	userepl	passwd	valbegin	valuntil
Ruby	10	t	t	t	t	*****		default_pool
dbadmin	16393	f	f	f	f	*****		default_pool
jack	451897	f	f	f	f	*****		default_pool
emma	451910	f	f	f	f	*****		default_pool
jerry	457386	f	f	f	f	*****	2022-05-19 10:31:56+08	default_pool

(5 rows)

### Method 2:

Check the **passwordtime** column in the **PG\_AUTH\_HISTORY** system catalog. This column indicates the time when the user's initial password was created. Only users with system administrator permissions can access the catalog.

```
select roloid, min(passwordtime) as create_time from pg_auth_history group by roloid order by roloid;
```

The following is an example:

Query the **PG\_USER** view to obtain the OID of user **jerry**, which is **457386**. Query the **passwordtime** column to obtain the creation time of user **jerry**, which is **2022-05-19 10:31:56**.

```
select roloid, min(passwordtime) as create_time from pg_auth_history group by roloid order by roloid;
```

10		2022-02-25 09:53:38.711785+08
16393		2022-02-25 09:55:17.992932+08
451897		2022-05-18 09:42:26.897855+08
451910		2022-05-18 09:46:33.152354+08
457386		2022-05-19 10:31:56.037706+08
(5 rows)		

## 1.9 Regions and AZs

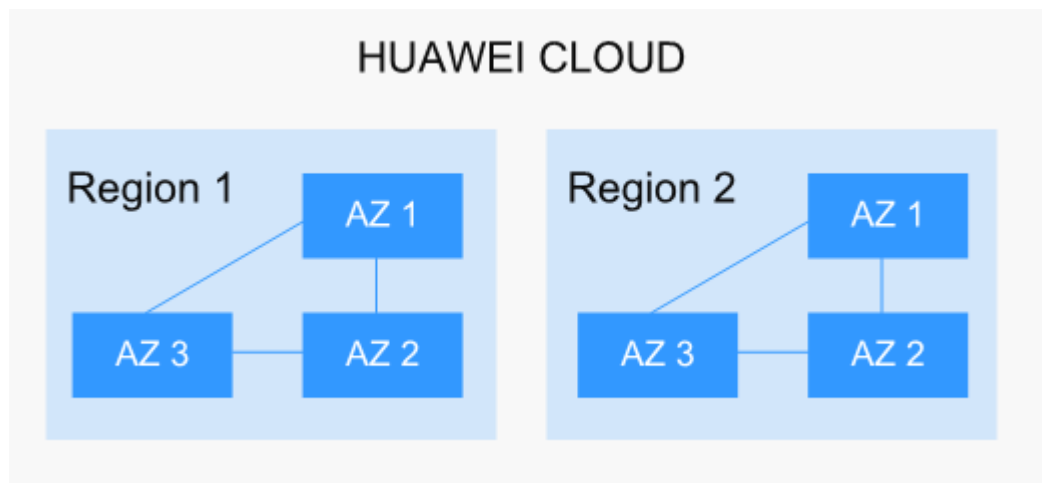
### Concepts

A region and availability zone (AZ) identify the location of a data center. You can create resources in regions and AZs.

- **Regions** are defined in terms of geographical location and network latency. Each region has its own shared public services (ECS, EVS, OBS, VPC, EIP, and IMS). Regions are either common or dedicated. A common region provides common cloud services available to all tenants. A dedicated region provides services of a specific type or only for specific tenants.
- An **AZ** contains one or more physical data centers. Each AZ has independent cooling, fire extinguishing, antimoisture, and electricity facilities. The computing, network, storage, and other resources in an AZ are logically divided into multiple clusters. AZs in a region are interconnected through high-speed optic fiber, so systems deployed across AZs can achieve higher availability.

**Figure 1-1** shows the relationship between the regions and AZs.

**Figure 1-1** Regions and AZs



### How Do I Select a Region?

When selecting a region, consider the following:

- **Location**  
Select a region close to you or your target users for lower network latency and faster access. Regions in the Chinese mainland provide the same infrastructure, BGP network quality, and operations and configurations on

resources. Therefore, if your target users are in the Chinese mainland, you do not need to consider network latency differences when selecting a region.

Countries and regions outside the Chinese mainland, such as Bangkok and Hong Kong, provide services for users outside the Chinese mainland. If you or your target users are in the Chinese mainland, these regions are not recommended due to high access latency.

- If you or your target users are in the Asia Pacific region (excluding the Chinese mainland), select the **AP-Bangkok** or **AP-Singapore** region.
- If you or your target users are in Africa, select the **AF-Johannesburg** region.
- If you or your target users are in Europe, select the **EU-Paris** region.
- Resource pricing  
This varies by region. For details, see the [pricing details](#).

## How Do I Select an AZ?

Consider your requirements for DR and network latency when selecting an AZ:

- Deploy resources in different AZs in the same region for DR purposes.
- Deploy resources in the same AZ for minimum latency.

## Regions and Endpoints

When you use resources with API calls, you must specify the regional endpoint. Obtain the regions and endpoints from the enterprise administrator.

## 1.10 Is My Data Secure in GaussDB(DWS)?

Yes. In the big data era, data has become a core asset. The public cloud will adhere to the commitment made over the years that we do not touch your applications or data, helping you protect your core assets. This is our commitment to users and the society, laying the foundation for the business success of the public cloud and their partners.

GaussDB(DWS) is a data warehousing system with telecom-class security to safeguard your data and privacy. Moreover, the public cloud GaussDB(DWS) delivers carrier-class quality, which can satisfy data security and privacy requirements of governments, financial organizations, and carriers. Therefore, it is widely used by various industries. GaussDB(DWS) of the public cloud won the following security authentication:

- Internal Cyber Security Lab (ICSL) in compliance with cyber security standards issued by the UK authorities.
- Privacy and Security Assessment (PSA) to meet EU requirements of data security and privacy.

## Service Data Security

GaussDB(DWS) is built on public cloud software infrastructure, including ECS and OBS. Both ECS and OBS won the Trusted Cloud Certification issued by the China Data Center Alliance in 2017.

Service data of GaussDB(DWS) users is stored in the ECSs in the cluster. Neither users nor public cloud O&M administrators can log in to the ECSs.

The operating system of ECSs is hardened for security, including kernel hardening, installation of the latest patch, permission control, port management, and protocol and port anti-attack.

GaussDB(DWS) provides complete security measures, such as password policies, authentication, session management, user permissions management, and database audit.

## Snapshot Data Security

GaussDB(DWS) backups are snapshots stored in OBS. OBS has passed the China Data Center Alliance trusted cloud security authentication. OBS supports access permission control, key access, and data encryption features. GaussDB(DWS) snapshot data can be used for data backup and restoration only and cannot be accessed by any user. GaussDB(DWS) administrators can view the OBS space occupied by snapshot data on the GaussDB(DWS) console and public cloud bills.

## Network Access Security

GaussDB(DWS) is fully isolated between the layer-2 and layer-3 networks to fulfill security requirements of government and financial users.

- GaussDB(DWS) is deployed in the tenant-dedicated ECS environment, which is not shared with other tenants. Therefore, data leakage due to computing resource sharing is impossible physically.
- ECSs in a GaussDB(DWS) cluster are isolated through VPCs, preventing the ECSs from being discovered and intruded on by other tenants.
- The network is divided into the service plane and management plane. The two planes are physically isolated, ensuring network security.
- The tenants can flexibly customize the security group and access rules.
- External application software access GaussDB(DWS) over SSL.
- Data imported from OBS is encrypted.

### 1.11 How Is GaussDB(DWS) Secured?

GaussDB(DWS) uses IAM and VPC to control user access and isolate cluster network. Cluster access is over SSL and cipher suite. Additionally, GaussDB(DWS) supports two-way digital certificate authentication.

Node OSs in each cluster are hardened to allow valid access to only OS files.

### 1.12 Can I Modify the Security Group of a GaussDB(DWS) Cluster?

Yes. When a data warehouse cluster is created, its security group cannot be changed. However, you can add, delete, or modify rules of the current security group.

To edit the cluster security group:

1. Log in to the GaussDB(DWS) management console.
2. In the navigation pane on the left, click **Clusters**.
3. In the cluster list, find the target cluster and click the cluster name. The **Basic Information** page is displayed.
4. Locate the **Security Group** parameter and click the security group name to switch to the **Security Groups** page on the VPC console, on which you can set the security group.

## 1.13 How Are LibrA, GaussDB A, and GaussDB(DWS) Related?

GaussDB(DWS) is an online data processing database built on the public cloud infrastructure and platform. It evolved from Huawei's own GaussDB A (originally called FusionInsight LibrA). GaussDB A is a database software deployed on physical machines. For more information, visit the following websites:

- Version 6.5.1 or earlier: <https://support.huawei.com/enterprise/en/cloud-computing/gaussdb-200-pid-21407429>
- Version 8.0.0: <https://support.huawei.com/enterprise/en/cloud-computing/gaussdb-a-pid-250949677>

## 1.14 What Is a Database/Data Warehouse/Data Lake/Lakehouse?

The evolving Internet and IoT produce massive volumes of data. This data needs to be managed, using concepts like database, data warehouse, data lake, and lakehouse. What are these concepts? What are their relationships? What are the specific products and solutions? This document helps you understand them through comparison.

### Database

A database is where data is organized, stored, and managed by data structure.

Databases have been used in computers since the 1960s, with the two prevailing data models (hierarchical and network), and data and applications were very interdependent. This limited database applications.

A database usually refers to a relational database. A relational database organizes data with a relational model, that is, data is stored in rows and columns. Therefore, database data is well-structured and independent with low redundancy. In 1970, relational databases were born to completely separate data from applications for software and have become an indispensable part of mainstream computer systems. Relational databases are the foundation of database products from all vendors, with relational API support even if a database is non-relational.

Relational databases process basic and routine transactions using OLTP, such as bank transactions.

## Data Warehouse

Database growth has facilitated data growth. OLAP explores the relationship between data and mines more data value. However, it is difficult to share data between different databases, and data integration and analysis also face great challenges.

To overcome these challenges for enterprises, Bill Inmon, proposed the idea of data warehousing in 1990. The data warehouse runs on a unique storage architecture to perform OLAP on a large amount of the OLTP data accumulated over the years. In this way, enterprises can obtain valuable information from massive data quickly and effectively to make informed decisions. Thanks to data warehouses, the information industry has evolved from operational systems based on relational databases to decision support systems.

Unlike a database, a data warehouse has the following features:

- A data warehouse uses themes. It is built to support various services, with data coming from scattered operational data. Therefore, the required data needs to be extracted from multiple heterogeneous data sources, processed and integrated, and reorganized by theme.
- A data warehouse mainly supports enterprise decision analysis and the operations involved are focused on data query. Therefore, it improves the query speed and cuts the total cost of ownership (TCO) by optimizing table structures and storage modes.

**Table 1-3** Comparison between data warehouses and databases

Dimension	Data Warehouse	Database
Application scenario	OLAP	OLTP
Data source	Multiple	Single
Data normalization	Denormalized schemas	Highly normalized static schemas
Data access	Optimized read operations	Optimized write operations

## Data Lake

Data is an important asset for enterprises. Production and operations data are saved and distilled into effective management policies.

The data lake does that. It is a large data warehouse that centrally stores structured and unstructured data. It can store raw data of multiple data sources and types, meaning that data can be accessed, processed, analyzed, and transmitted without being structured first. The data lake helps enterprises quickly complete federated analysis of heterogeneous data sources and explore data value.

A data lake is in essence a solution that consists of a data storage architecture and data processing tools.



- The **storage architecture** must be scalable and reliable enough to store massive data of any type (structured, semi-structured, unstructured data).
- The two types of **processing tools** have separate functions:
  - The first type: migrates data into the lake, including defining sources, formulating synchronization policies, moving data, and compiling catalogs.
  - The second type then uses that data, including analyzing, mining, and using it. The data lake must be equipped with wide-ranging capabilities, such as comprehensive data and data lifecycle management, diversified data analytics, and secure data acquisition and release. These data governance tools help guarantee data quality, which can be compromised by a lack of metadata and turn the data lake into a data swamp.

Now with big data and AI, lake data is even more valuable and plays new roles. It represents more enterprise capabilities. For example, the data lake can centralize data management, helping enterprises build more optimized operation models. It also provides other enterprise capabilities such as prediction analysis and recommendation models. These models can stimulate further growth.

Just like any other warehouse and lake, one stores goods, or data, from one source while the other stores water, or data, from many sources.

**Table 1-4** Comparison between data lakes and data warehouses

Dimension	Data Lake	Data Warehouse
Application scenario	Exploratory analytics (machine learning, data discovery, profiling, prediction)	Data analytics (based on historical structured data)
Cost	Low initial cost, high subsequent cost	High initial cost, low subsequent cost
Data quality	Massive raw data to be cleaned and normalized before use	High quality data that can be used as the basis of facts
Target user	Data scientists and data developers	Business analysts

## Lakehouse

Although the application scenarios and architectures of a data warehouse and a data lake are different, they can cooperate to resolve problems. A data warehouse stores structured data and is ideal for quick BI and decision-making support, while a data lake stores data in any format and can generate larger value by mining data. Therefore, their convergence can bring more benefits to enterprises in some scenarios.

A lakehouse, the convergence of a data warehouse and a data lake, aims to enable data mobility and streamline construction. The key of the lakehouse architecture is to enable the free flow of data/metadata between the data

warehouse and the data lake. The explicit-value data in the lake can flow to or even be directly used by the warehouse. The implicit-value data in the warehouse can also flow to the lake for long-term storage at a low cost and for future data mining.

## Intelligent Data Solution

DataArts Studio is a data enablement platform that helps large government agencies and companies customize intelligent data resource management solutions. This solution can import all-domain data into the data lake, eliminating data silos, unleashing the value of data, and empowering data-driven digital transformation.

DataArts Studio features the FusionInsight intelligent data lake as its core. Around it are computing engines such as the database, data warehouse, data lake, and data platform. It provides comprehensive data enablement, covering data collection, aggregation, computing, asset management, and data openness.

Lake, warehouse, and database engines enable agile data lake construction, fast migration of GaussDB databases, and real-time analysis of the data warehouse. For more information, go to:

- Database
  - Relational databases include: [Relational Database Service \(RDS\)](#) , , , [RDS for PostgreSQL](#) , .
  - Non-relational database: [Document Database Service \(DDS\)](#), GaussDB NoSQL (including Influx, Redis, Mongo, Cassandra)
- Data warehouse: [GaussDB\(DWS\)](#)
- Data lake and warehouse integration: [MMapReduce Service \(MRS\)](#), [Data Lake Insight \(DLI\)](#) .
- Data governance center: [DataArts Studio](#).

# 2 Cluster Management

---

## 2.1 What Do I Do If Creating a GaussDB(DWS) Cluster Failed?

### Troubleshooting

Check that you have enough quota for creating the cluster.

### Technical Support

If the fault cannot be identified, submit a service ticket to report the problem: Log in to the management console and choose **Service Tickets > Create Service Ticket**.

## 2.2 How Can I Clear and Reclaim the Storage Space?

After you delete data stored in GaussDB(DWS) data warehouses, dirty data may be generated possibly because the disk space is not released. This results in disk space waste and deteriorates snapshot creation and restoration performance. The following describes the impact on the system and subsequent operation to clear the disk space:

Points worth mentioning during clearing and reclaiming storage space:

- Unnecessary data needs to be deleted to release the storage space.
- Frequent read and write operations may affect proper database use. Therefore, it is good practice to clear and reclaim the storage space when not in peak hours.
- The data clearing time depends on the data stored in the database.

Perform the following steps to clear and reclaim the storage space:

1. Connect to the database. For details, see [Methods of Connecting to a Cluster](#).
2. Run the following command to clear and reclaim the storage space:

### **VACUUM FULL;**

By default, tables the current user has the permission on are deleted. Other tables are skipped.

The following information is displayed once the space is cleared:

VACUUM

#### **NOTE**

- **VACUUM FULL** reclaims all expired row space, however it requires an exclusive lock on each table being processed, and might take a long time to complete on large, distributed database tables. You are advised to do **VACUUM FULL** to specified tables. If you want to do **VACUUM FULL** to the entire database, you are advised to do it during database maintenance.
- The statistical information will be lost if you use the **FULL** parameter. To collect the statistics, add keyword **ANALYZE**, for example, **VACUUM FULL ANALYZE**;  
For more information about **VACUUM**, see [VACUUM](#) in the *SQL Syntax Reference*.

## 2.3 Can I Switch My Cluster Nodes to Another Region After Purchase?

No. Cluster nodes cannot be used across regions.

Instead, cancel the order in the original region, place a new order in the desired region, and create a cluster.

## 2.4 Why Did the Used Storage Shrink After Scale-out?

### Possible Causes

If you do not run **VACUUM** to clear and reclaim the storage space before the scale-out, the data deleted from GaussDB(DWS) may not free up the occupied disk space.

During the scale-out, the system redistributes the data because the service data volume on the original nodes is significantly larger than that on the newly added nodes. When the redistribution starts, the system automatically performs **VACUUM** to free up the storage space. This causes a big drop in capacity.

### Handling Procedure

You are advised to periodically clear and reclaim the storage space by running **VACUUM FULL** to prevent data expansion.

If the used storage space is still large after you run **VACUUM FULL**, analyze whether the existing cluster flavor meets service requirements. If no, scale out the cluster.

## 2.5 How Do I View Node Metrics (CPU, Memory, and Disk Usage)?

You can view the used capacity of a cluster CPU, memory, and disks on the Cloud Eye management console. Perform the following steps to view the information:

**Step 1** Log in to the GaussDB(DWS) console and click **Viewing Metric** next to a cluster.

**Step 2** Click  to return to the **Cloud Service Monitoring** page, switch to the **Data Warehouse Node** page, and click **View Metric** on the right of the corresponding node to view its disk usage.

----End

## 2.6 Does GaussDB(DWS) Support BMS?

Yes. You can submit a service ticket to apply for BMS flavors. GaussDB(DWS) uses ECSs only on HUAWEI CLOUD by default.

## 2.7 How Is the Disk Space or Capacity of GaussDB(DWS) Calculated?

1. Total disk capacity of GaussDB(DWS): For a three-node cluster, if each node is 320 GB, the total capacity is 960 GB. When 1 GB data is stored, GaussDB(DWS) stores 1 GB data on two nodes due to duplication, a security mechanism, thereby occupying a total of 2 GB space. As a result, more than 2 GB space is occupied if metadata and indexes are calculated. Therefore, a three-node cluster with a total capacity of 960 GB can store 480 GB data. This mechanism ensures data security.

When you nodes on the console, you are billed by the available capacity of a node. For example, the actual space of **dws.m3.xlarge** is 320 GB and the available space displayed is 160 GB, the space you will be billed for.

2. Check the disk usage of a single node.

Similarly, if the total capacity is 960 GB and there are three data nodes, the disk capacity of each node is 320 GB.

Log in to the Gauss(DWS) console and choose **Monitoring > Node Monitoring > Overview** to view the usage of disks and other resources on each node.

### NOTE

- The disk space displayed on the **Node Management** page is the total capacity of all disks, that is, system disks and data disks, in the data warehouse cluster. The disk space displayed on the **Overview** page is only the available space for storing table data in the cluster. In addition, tables in the data warehouse cluster have backup copies, these copies also occupy the disk storage.
- If the cluster is read-only and an alarm for insufficient disk space is generated, expand the cluster capacity by following the instructions provided in [Scaling Out a Cluster](#).

## 2.8 What Are the gaussdb and postgres Databases of GaussDB(DWS)?

The **gaussdb** and **postgres** databases are built-in databases of GaussDB(DWS). You can create schemas and tables in them. However, you are advised to recreate a database and create schemas and tables in the new database.


## 2.9 How Do I Set the Maximum Number of Sessions When Adding an Alarm Rule on Cloud Eye?

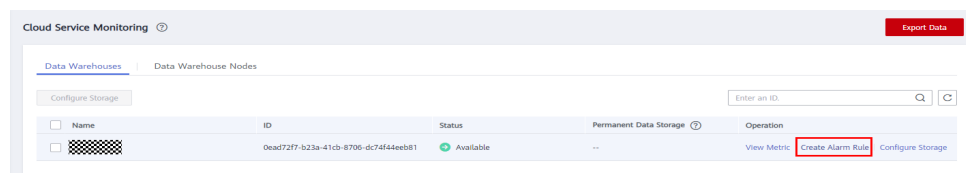
After connecting to a database, run the following SQL statement to check the maximum number of concurrent sessions globally:

```
show max_active_statements;
```

Go to the Cloud Eye console and set the threshold to 70% to 80% of the obtained value. For example, if the value of **max\_active\_statements** is **80**, set the threshold to **56** ( $80 \times 70\%$ ).

Procedure:

1. Go to the **Clusters** page on the GaussDB(DWS) management console.
2. Click **View Metric** in the **Operation** column of the target cluster to go to the Cloud Eye console.
3. Click  in the upper left corner on the displayed page and click **Create Alarm Rule** of the target cluster.



4. Set **Method** to **Configure manually**, **Metric Name** to **Session Count**, **Alarm Policy** to **56**, and **Alarm Severity** to **Major**. Then click **Create**.

The screenshot shows the configuration page for a Data Warehouse Service alarm. Key elements include:

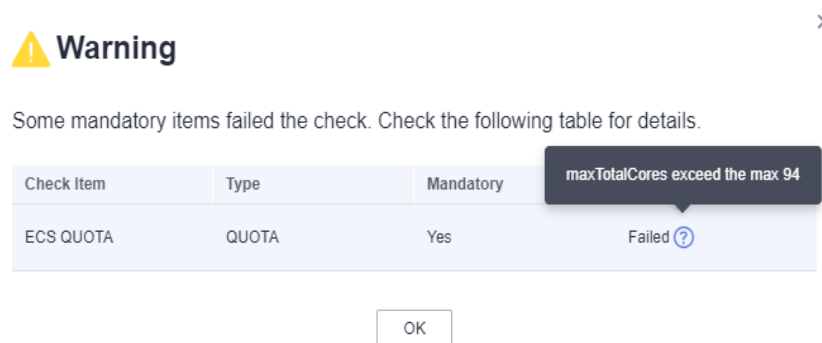
- Name:** alarm-hjvd
- Description:** (Empty text box, 0/256 characters)
- Enterprise Project:** default
- Resource Type:** Data Warehouse Service
- Dimension:** Data Warehouses
- Monitoring Scope:** Specific resources
- Monitored Object:** (Checkered icon)
- Method:** 'Configure manually' is selected.
- Alarm Policy Table:**

Metric Name	Alarm Policy	Alarm Severity	Operation
Session Count	Raw d...   3 consecuti...   >=   56   One day	Major	

## 2.10 What Should I Do If the Scale-out Check Fails?

### Symptom

After you click **OK** during scale-out or adding idle nodes, a warning message is displayed and you cannot go to the next step.



### Cause Analysis

Before submitting a scale-out task, the system checks items such as resource quotas and IAM permissions. Abnormal items will stop scale-out tasks from being submitted to prevent scale-out failures.

### Solution

- If the quota check fails, check whether the resource quota is sufficient. If the available nodes are insufficient, click **Increase Quota** to submit a work order to increase the node quota.
- IAM permissions: This problem occurs when a user has only an IAM account. In this case, permissions such as VPC and EVC/BMS permissions need to be

assigned to the IAM account. Or, the user can use the master account to perform scaling out.

- If there are no abnormal items but the dialog box still exists, contact technical support.

## 2.11 When Should I Add CNs or Scale out a cluster?

### Introduction to CN Concurrency

CN is short for Coordinator Node. A CN is an important component of GaussDB(DWS) and is closely related to users. It provides interfaces to external applications, optimizes global execution plans, distributes execution plans to DataNodes, and summarizes and processes execution results. A CN is an interface to external applications. The concurrency capability of the CN determines the service concurrency.

CN concurrency is determined by the following parameters:

- **max\_connections**: specifies the maximum number of concurrent connections to the database. This parameter affects the concurrent processing capability of the cluster. The default value depends on the cluster specifications. For details, see [Managing Database Connections](#).
- **max\_active\_statements**: specifies the maximum number of concurrent jobs. This parameter applies to all the jobs on one CN. The default value is **60**, which indicates a maximum of 60 jobs can run at the same time. Other jobs will be queued.

### Add CNs or Scale out a Cluster?

- **Insufficient connections**: When a cluster is created for the first time, the default number of CNs in the cluster is 3, which can meet the customer's basic connection requirements. If the cluster has a large number of concurrent requests and the number of connections to each CN is large, or the CPU usage of a CN exceeds its capacity, you are advised to add CNs. For details, see [CNs](#).
- **Insufficient storage capacity and performance**: If your business grows and you have higher requirements on storage capacity and performance, or the CPU of your cluster is insufficient, you are advised to scale out your cluster. For details, see [Scaling Out a Cluster](#).

With the expansion of cluster nodes, more CNs are needed to meet the distribution requirements of GaussDB(DWS). In short, adding CNs does not necessarily require cluster scale-out. However, after cluster scale-out, CNs may need to be added.

## 2.12 What Are the Scenarios of Resizing a Cluster, Changing the Node Flavor, Scale-out, and Scale-in?

Resizing a cluster has a great impact on your workloads. It is similar to migrating an old cluster to a new one, with changes on nodes and specifications. You are



advised to perform lightweight operations, such as scale-out, scale-in, and flavor change. The following table lists the application scenarios of the cluster modification options.

**Table 2-1** Cluster modification options

Option	Applicable Scenario	Remarks
Scale-out	If your business grows and you have higher requirements on storage capacity and performance, or the CPU of your cluster is insufficient, you are advised to scale out your cluster.	Nodes cannot be added to a hybrid data warehouse (standalone).
Scale-in	During off-peak hours when a large amount of cluster capacity is idle, you can reduce the number of nodes to reduce costs.	A hybrid data warehouse (cluster mode) cannot be scaled in to a standalone cluster.
Changing the Node Flavor	This option changes cluster flavors (including CPU, memory, and others) to meet service requirements. It does not change the number of nodes.	Currently, you can only change the flavors of cloud data warehouse clusters and stream data warehouse clusters that only use ECS and EVS resources for computing and storage.
Changing all specifications	You can resize your cluster when: <ul style="list-style-type: none"><li>• Your clusters are BMS clusters or they do not support flavor change.</li><li>• You want to change the cluster topology instead of scale-out or scale-in which simply adds nodes or delete nodes ring by ring.</li><li>• Your cluster is aged and you want to change it to a new cluster without migrating data.</li></ul>	Currently, only standard data warehouse clusters are supported.

## 2.13 How Should I Select from a Small-Flavor Many-Node Cluster and a Large-Flavor Three-Node Cluster with Same CPU Cores and Memory?

- **Small-flavor many-node:**  
If your data volume is small and you have requirement for node scaling, but you have limited budget, you can select a small-flavor many-node cluster.  
For example, a small-flavor cluster (dwsx2.h.2xlarge.4.c6) with 8 cores and 32 GB memory can provide strong computing capabilities. The cluster has a large number of nodes and can process high concurrent requests. In this case, you only need to ensure that the network speed between nodes is normal to avoid cluster performance limitation.
- **Large-flavor three-node:**  
If you have a large amount of data to be processed, have high requirement on computing, and have a high budget, you can select a large-flavor three-node cluster.  
For example, a large-flavor cluster (dws2.m6.8xlarge.8) with 32 cores and 256 GB memory has faster CPU processing capability and larger memory, and can process data more quickly. However, the cluster has limited nodes, which may cause low performance in high-concurrency scenarios.

## 2.14 What Are the Differences Between Cloud SSDs and Local SSDs?

Cloud SSDs support scale-out. Therefore, you are advised to use them. The differences between cloud SSDs and local SSDs are as follows:

- **Cloud SSDs**
  - Cloud SSDs use EVS as storage media and can be scaled out as data grows. This is more flexible.
  - Cloud SSDs are not bound to the ECS flavor. Therefore, you can adjust the flavors of cloud SSDs when needed.
- **Local SSDs**
  - Local SSDs use ECS local disks as storage media. They have fixed capacity and higher performance but cannot be scaled out.
  - If the capacity is insufficient, you have to add more nodes to increase capacity. The flavors of local SSDs cannot be adjusted.

## 2.15 What Are the Differences Between Hot Data Storage and Cold Data Storage?

The biggest difference between hot data storage and cold data storage lies in the storage media.

- Hot data is frequently queried or updated and has high requirements on access response time. It is stored on **DN data disks**.
- Cold data is not updated and is occasionally queried, and does not have high requirements on access response time. It is stored in **OBS**.

Different storage media determine the cost, performance, and application scenarios of the two storage mode, as shown in [Table 2-2](#).

**Table 2-2** Differences between hot and cold data storage

Storage	Read and Write	Cost	Capacity	Application Scenario
Hot storage	Fast	High	Fixed and restricted	This mode is applicable to scenarios where the data volume is limited and needs to be frequently read and updated.
Cold storage	Slow	Low	Large and unlimited	This mode is applicable to scenarios such as data archiving. It features low cost and large capacity.

# 3 Database Connections

## 3.1 How Applications Communicate with GaussDB(DWS)?

For applications to communicate with GaussDB(DWS), make sure the networks between them are connected. The following table lists common connection scenarios.

**Table 3-1** Communication between applications and GaussDB(DWS)

Scenario		Description	Supported Connection Type
Cloud	<a href="#">Service Application and GaussDB(DWS) Are in the Same VPC in the Same Region</a>	Two private IP addresses in the same VPC can directly communicate with each other.	<ul style="list-style-type: none"><li>• gsql</li><li>• Data Studio</li><li>• JDBC/ODBC</li></ul> For more connection modes, see <a href="#">Methods of Connecting to a Cluster</a> .
	<a href="#">Service Applications and GaussDB(DWS) are in Different VPCs in the Same Region</a>	After a <b>VPC peering</b> connection is created between two VPCs, the two private IP addresses can directly communicate with each other.	
	<a href="#">Service Applications and GaussDB(DWS) Are in Different Regions</a>	After a <b>cloud connection</b> (CC) is established between two regions, the two regions communicate with each other through private IP addresses.	

Scenario		Description	Supported Connection Type
On-premises and on-cloud	<b>Service applications are deployed in on-premise data centers and need to communicate with GaussDB(DWS).</b>	<ul style="list-style-type: none"> <li>Use the <b>public IP address or domain name</b> of GaussDB(DWS) for communication.</li> <li>Use <b>Direct Connect (DC)</b> is for communication.</li> </ul>	

## Service Application and GaussDB(DWS) Are in the Same VPC in the Same Region

To ensure low service latency, you are advised to deploy service applications and GaussDB(DWS) in the same region. For example, if a service application is deployed on an ECS, you are advised to deploy the data warehouse cluster in the same VPC as the ECS. In this way, the application can directly communicate with GaussDB(DWS) through an intranet IP address. In this case, deploy the data warehouse cluster in the same region and VPC where the ECS resides.

For example, if the ECS is deployed in **EU-Dublin**, select **EU-Dublin** for the GaussDB(DWS) cluster and ensure that the GaussDB(DWS) cluster and the ECS are both in **VPC1**. The private IP address of the ECS is **192.168.120.1**, the private IP address of GaussDB(DWS) is **192.168.120.2**. Therefore, they can communicate with each other through private IP addresses.

The key points in communication check are the ECS outbound rule and GaussDB(DWS) inbound rule. The check procedure is as follows:

### Step 1 Check the ECS outbound rules:

Ensure that the outbound rule of the ECS security group allows access. If access is not allowed, see the [Configuring Security Group Rules](#).

Action	Protocol & Port	Type	Destination
Allow	All	IPv4	0.0.0.0/0

### Step 2 Check the GaussDB(DWS) inbound rules:

If no security group is configured when GaussDB(DWS) is created, the default inbound rule allows TCP access from all IPv4 addresses and port 8000. To ensure security, you can also allow only one IP address. For details, see [How Do I Configure a Whitelist to Protect Clusters Available Through a Public IP Address?](#)

Allow	TCP : 8000	IPv4	0.0.0.0/0
-------	------------	------	-----------

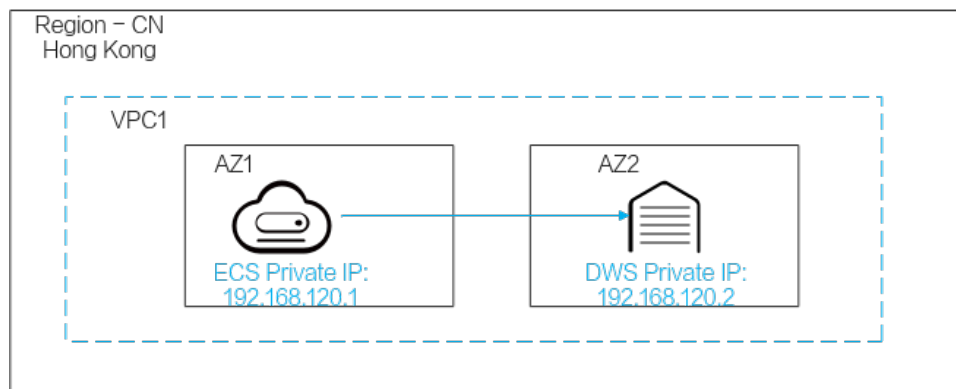
**Step 3** Log in to the ECS. If the internal IP address of GaussDB(DWS) can be pinged, the network connection is normal. If the IP address cannot be pinged, check the preceding configuration. If the ECS has a firewall, check the firewall configuration.

----End

**Example of using gsql for connection:**

```
gsql -d gaussdb -h 192.168.120.2 -p 8000 -U dbadmin -W password -r
```

**Figure 3-1** Access via Private IP addresses



## Service Applications and GaussDB(DWS) are in Different VPCs in the Same Region

To ensure low service latency, you are advised to deploy service applications and GaussDB(DWS) in the same region. For example, if service applications are deployed on an ECS, you are advised to deploy the data warehouse cluster in the same VPC as the ECS. If a different VPC is selected for the data warehouse cluster, the ECS cannot directly connect to GaussDB(DWS).

For example, both ECS and GaussDB(DWS) are deployed in **EU-Dublin**, but ECS is in VPC1 and GaussDB(DWS) is in VPC2. In this case, you need to create a **VPC Peering Connection** between VPC1 and VPC2 so that ECS can access GaussDB(DWS) using the private IP address of GaussDB(DWS).

The key points for checking the communication are the ECS outbound rules, GaussDB(DWS) inbound rules, and VPC peering connection. The check procedure is as follows:

### Step 1 Check the ECS outbound rules:

Ensure that the outbound rule of the ECS security group allows access. If access is not allowed, see the [Configuring Security Group Rules](#).

Action ?	Protocol & Port ?	Type	Destination ?
Allow	All	IPv4	0.0.0.0/0 ?

### Step 2 Check the GaussDB(DWS) inbound rules:

If no security group is configured when GaussDB(DWS) is created, the default inbound rule allows TCP access from all IPv4 addresses and port 8000. To ensure

security, you can also allow only one IP address. For details, see [How Do I Configure a Whitelist to Protect Clusters Available Through a Public IP Address?](#)

Allow	TCP : 8000	IPv4	0.0.0.0/0 ?
-------	------------	------	-------------

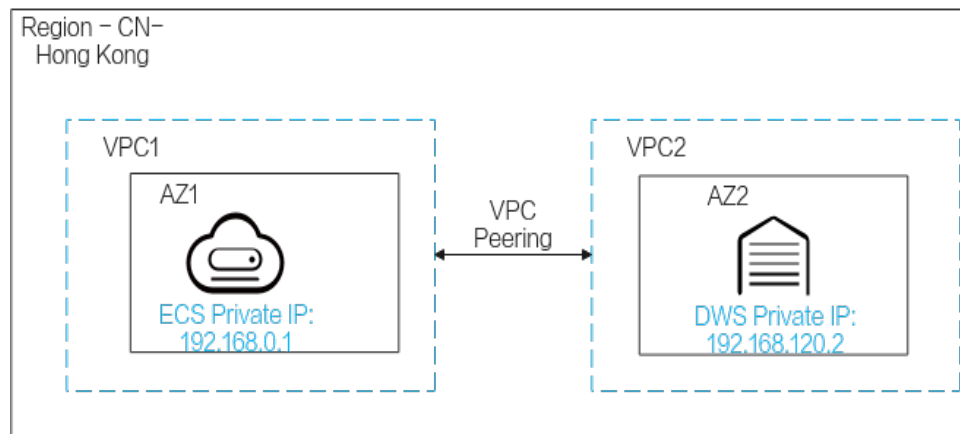
- Step 3** Create a **VPC Peering Connection** between VPC1 where the ECS is and VPC2 where GaussDB(DWS) is.
- Step 4** Log in to the ECS. If the internal IP address of GaussDB(DWS) can be pinged, the network connection is normal. If the IP address cannot be pinged, check the preceding configuration. If the ECS has a firewall, check the firewall configuration.

----End

**Example of using gsql for connection:**

```
gsql -d gaussdb -h 192.168.120.2 -p 8000 -U dbadmin -W password -r
```

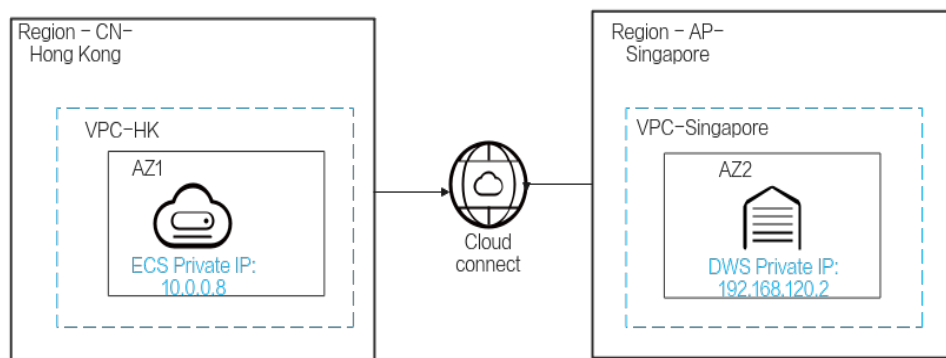
**Figure 3-2** Access via VPC peering



### Service Applications and GaussDB(DWS) Are in Different Regions

If the service application and GaussDB(DWS) are in different regions, for example, ECS is in **EU-Dublin** and GaussDB(DWS) is in another region, you need to establish a **Cloud Connect** between the two regions for communication.

**Figure 3-3** Access via cloud connect



### Service applications are deployed in on-premise data centers and need to communicate with GaussDB(DWS).

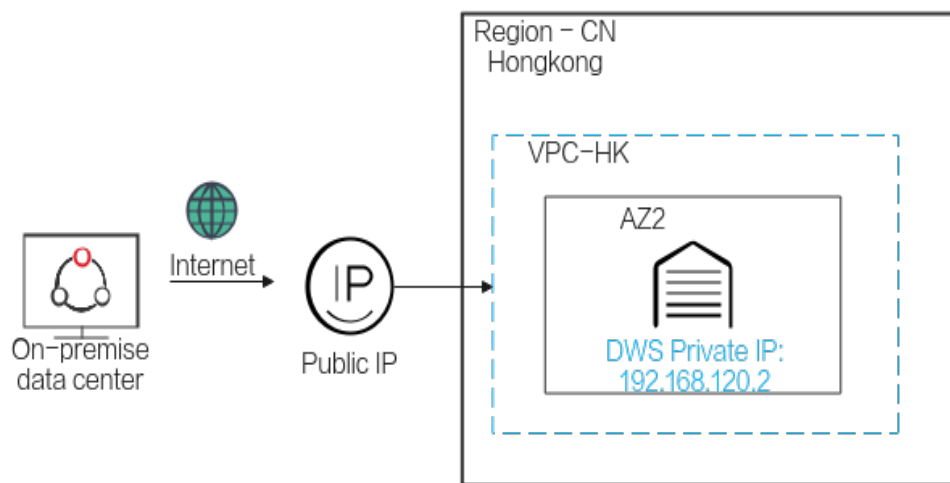
If service applications are not on the cloud but in the local data center, they need to communicate with GaussDB(DWS) on the cloud.

- **Scenario 1:** On-premises service applications communicate with GaussDB(DWS) through GaussDB(DWS) public IP addresses.

Example of using gsql for connection:

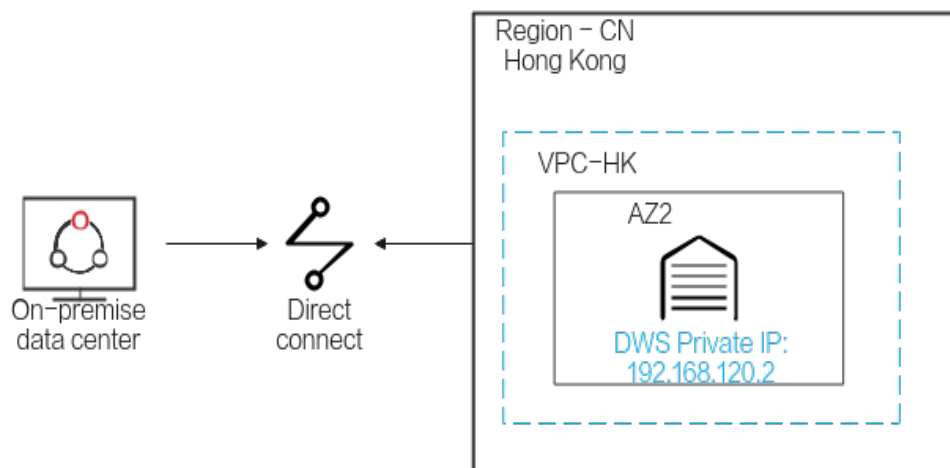
```
gsql -d gaussdb -h public_IP_address -p 8000 -U dbadmin -W password -r
```

Figure 3-4 Access via public IP addresses



- **Scenario 2:** On-premises services cannot access the external network. In this case, **Direct Connect** is required for communication.

Figure 3-5 Access via direct connect





## 3.2 Does GaussDB(DWS) Support Third-Party Clients and JDBC and ODBC Drivers?

Yes, but GaussDB(DWS) clients and drivers are recommended. Unlike open-source PostgreSQL clients and drivers, GaussDB(DWS) clients and drivers have two key advantages:

- **Security hardening:** PostgreSQL drivers only support MD5 authentication, but GaussDB(DWS) drivers support SHA256 and MD5.
- **Data type enhancement:** GaussDB(DWS) drivers support new data types `smalldatetime` and `tinyint`.

GaussDB(DWS) supports open-source PostgreSQL clients and JDBC and ODBC drivers.

The compatible client and driver versions are:

- PostgreSQL `psql` 9.2.4 or later
- PostgreSQL JDBC Driver 9.3-1103 or later
- PSQL ODBC 09.01.0200 or later

For details about how to use JDBC/ODBC to connect to GaussDB(DWS), see [Guide: JDBC- or ODBC-Based Development](#).

## 3.3 Can I Connect to GaussDB(DWS) Cluster Nodes Using SSH?

No, direct access is not supported. VMs at the bottom layer of GaussDB(DWS) serve as the compute nodes for data analysis. Access cluster databases using the private or public network access address instead.

## 3.4 What Should I Do If I Cannot Connect to a Data Warehouse Cluster?

### Troubleshooting

Check:

- Whether the cluster status is normal.
- Whether the connection command, username, password, IP address, and port are correct.
- Whether the operating system type and version of the client are correct.
- Whether the client is properly installed.

If cluster connection failed on the public cloud, check the following items:

- Whether the ECSs are in the same AZ, VPC, subnet, and security group as the cluster.

- Whether the inbound and outbound rules of the security group are correct.

If cluster connection failed through the Internet, confirm the following items:

- Whether your network is connected to the Internet.
- Whether the firewall blocked access.
- Whether you need to access the Internet through a proxy.

## Technical Support

If the fault cannot be identified, submit a service ticket to report the problem: Log in to the management console and choose **Service Tickets > Create Service Ticket**.

### 3.5 Why Was I Not Notified of Failure Unbinding the EIP When GaussDB(DWS) Is Connected Over the Internet?

After the EIP is unbound, the network may be disconnected. However, the TCP layer does not detect a faulty physical connection in time due to keepalive settings. As a result, the gsql, ODBC, and JDBC clients also cannot identify the network fault in time.

The duration when the database sends the disconnection message to the client depends on the keepalive settings. The specific algorithm for calculating the duration is:

**keepalive\_time + keepalive\_probes × keepalive\_intvl**

Keepalive values affect network communication stability. Adjust them to service pressure and network conditions.

On Linux, run the **sysctl** command to modify the following parameters:

- net.ipv4.tcp\_keepalive\_time
- net.ipv4.tcp\_keealive\_probes
- net.ipv4.tcp\_keepalive\_intvl

For example, if you want to change the value of **net.ipv4.tcp\_keepalive\_time**, run the following command to change it to **120**.

***sysctl net.ipv4.tcp\_keepalive\_time=120***

On Windows, modify the following configuration information in registry **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters**:


- KeepAliveTime
- KeepAliveInterval
- TcpMaxDataRetransmissions (equivalent to **tcp\_keepalive\_probes**)

 NOTE

If you cannot find the preceding parameters in registry **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters**, add these parameters. Open **Registry Editor**, right-click the blank area on the right, and choose **Create > DWORD (32-bit) Value** to add these parameters.

## 3.6 How Do I Configure a Whitelist to Protect Clusters Available Through a Public IP Address?

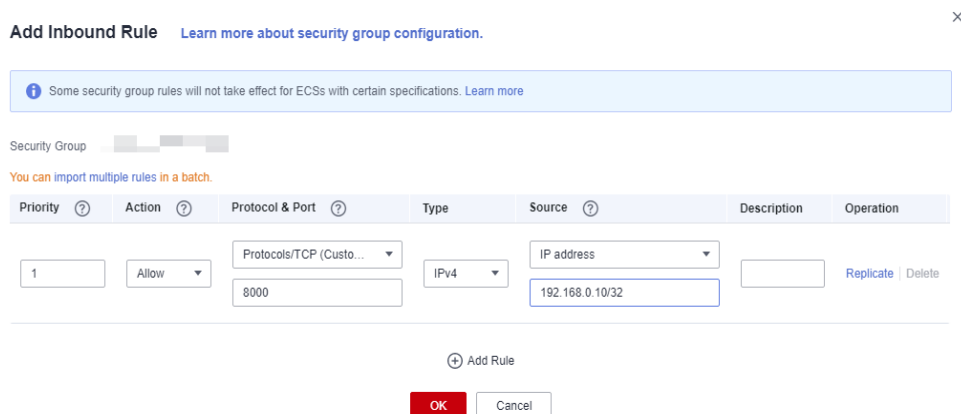
You can also log in to the [VPC management console](#) to manually create a security group. Then, go back to the page for creating data warehouse clusters,

click the  button next to the **Security Group** drop-down list to refresh the page, and select the new security group.

To enable the GaussDB(DWS) client to connect to the cluster, you need to add an inbound rule to the new security group to grant the access permission to the database port of the GaussDB(DWS) cluster.

- **Protocol: TCP**
- **Port: 8000** Use the database port set when creating the GaussDB(DWS) cluster. This port is used for receiving client connections to GaussDB(DWS).
- **Source:** Select **IP address** and use the host IP address of the client host, for example, **192.168.0.10/32**.

**Figure 3-6** Adding an inbound rule



Add Inbound Rule [Learn more about security group configuration.](#)

*Some security group rules will not take effect for ECSs with certain specifications. [Learn more](#)*

Security Group: [Redacted]

You can import multiple rules in a batch.

Priority	Action	Protocol & Port	Type	Source	Description	Operation
1	Allow	Protocols/TCP (Custom) 8000	IPv4	IP address 192.168.0.10/32		Replicate   Delete

[+](#) Add Rule

**OK** Cancel

The whitelist will be added.

# 4 Data Import and Export

---

## 4.1 What Are the Differences Between Data Formats Supported by OBS and GDS Foreign Tables?

The file formats supported by OBS and GDS foreign tables are as follows:

OBS supports ORC, TEXT, JSON, CSV, CARBONDATA and PARQUET file formats for data import and ORC, CSV, and TEXT file formats for data export. The default format is TEXT.

GDS supports the following file formats: TEXT, CSV, and FIXED. The default format is TEXT.

## 4.2 How Do I Import Incremental Data Using an OBS Foreign Table?

When you use an OBS foreign table to import data, **INSERT** imports the data to a local physical table. When OBS data is updated, you do not need to run the **INSERT** statement again. You can use the **MERGE INTO** statement.

## 4.3 How Can I Import Data to GaussDB(DWS)?

GaussDB(DWS) supports efficient data import from multiple data sources. The following lists typical data import modes. For details, see section [Importing Data](#) in the *Data Warehouse Service (DWS) Developer Guide*

- Importing data from the OBS  
Upload data to OBS and then export it to GaussDB(DWS) clusters. Data formats such as CSV and TEXT are supported.
- Inserting data with **INSERT** statements  
Use the `gsql` client tool provided by GaussDB(DWS) or the JDBC/ODBC driver to write data to GaussDB(DWS) from upper-layer applications. GaussDB(DWS) supports complete database transaction-level CRUD

operations. This is the simplest method and is applicable to scenarios with small data volume and low concurrency.

- Importing data from MRS with MRS as the ETL.
- Importing data with the **COPY FROM STDIN** command  
Run the **COPY FROM STDIN** command to write data to a table.
- Importing data from a remote server to GaussDB(DWS) using GDS  
Use the GDS data import function provided by GaussDB(DWS) to import data files from a common file system (for example, an ECS).
- Migrating data to GaussDB(DWS) using CDM

## 4.4 How Much Service Data Can a Data Warehouse Store?

Each node in a data warehouse cluster has a default storage capacity of 1.49 TB, 2.98 TB, 4.47 TB, 160 GB, 1.68 TB, or 13.41 TB. A cluster can house 3 to 256 nodes and the total storage capacity of the cluster expands proportionally as the cluster scale grows.

To enhance reliability, each node has a copy, which occupies half of the storage space.

The GaussDB(DWS) system backs up data and generates indexes, temporary cache files, and run logs, which occupy storage space. Therefore, the actual storage space of each node is about half of the total storage capacity.

## 4.5 How Do I Use \Copy to Import and Export Data?

GaussDB(DWS) is a fully managed service on the cloud. Users cannot log in to the background to import or export data by using **COPY**, so the **COPY** syntax is disabled. You are advised to store data files on OBS and use OBS foreign tables to import data. If you want to use **COPY** to import and export data, perform the following operations:

1. Place the data file on the client.
2. Use `gsq` to connect to the target cluster.
3. Run the following command to import data. Enter the directory name and file name of the data file on the client and specify the import option in **with**. The command is almost the same as the common **COPY** command. You only need to add a backslash (\) before the command. When the data is successfully imported, no notification will be displayed.  

```
\copy tb_name from '/directory_name/file_name' with(...);
```
4. Run the following command to export data to a local file. Retain the default settings of parameters.  

```
\copy table_name to '/directory_name/file_name';
```
5. Specify the **copy\_option** parameter to export data to a CSV file.  

```
\copy table_name to '/directory_name/file_name' CSV;
```
6. Use **with** to specify parameters, exporting data as CSV files that use vertical bars (|) as delimiters.  

```
\copy table_name to '/directory_name/file_name' with(format 'csv',delimiter '|') ;
```

## 4.6 How Do I Implement Fault Tolerance Import Between Different Encoding Libraries

To import data from database A (UTF8) to database B (GBK), there may be a character set mismatch error which causes the data import to fail.

To import a small amount of data, run the `\COPY` command. The procedure is as follows:

- Step 1** Create databases A and B. The encoding format of database A is UTF8, and that of database B is GBK.

```
postgres=> CREATE DATABASE A ENCODING 'UTF8' template = template0;
postgres=> CREATE DATABASE B ENCODING 'GBK' template = template0;
```

- Step 2** View the database list. You can view the created databases A and B.

```
postgres=> \l
                List of databases
  Name  | Owner  | Encoding | Collate | Ctype | Access privileges
-----+-----+-----+-----+-----+-----
 a      | dbadmin | UTF8     | C       | C     |
 b      | dbadmin | GBK      | C       | C     |
 gaussdb | Ruby   | SQL_ASCII | C       | C     |
 postgres | Ruby   | SQL_ASCII | C       | C     |
 template0 | Ruby   | SQL_ASCII | C       | C     | =c/Ruby      +
         |         |         |         | Ruby=CTc/Ruby
 template1 | Ruby   | SQL_ASCII | C       | C     | =c/Ruby      +
         |         |         |         | Ruby=CTc/Ruby
 xiaodi  | dbadmin | UTF8     | C       | C     |
(7 rows)
```

- Step 3** Switch to database A and enter the user password. Create a table named **test01** and insert data into the table.

```
postgres=> \c a
Password for user dbadmin:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_128_GCM_SHA256, bits: 128)
You are now connected to database "a" as user "dbadmin".

a=> CREATE TABLE test01
(
  c_customer_sk      integer,
  c_customer_id      char(5),
  c_first_name       char(6),
  c_last_name        char(8)
)
with (orientation = column,compression=middle)
distribute by hash (c_last_name);
CREATE TABLE
a=> INSERT INTO test01(c_customer_sk, c_customer_id, c_first_name) VALUES (3769, 'hello', 'Grace');
INSERT 0 1
a=> INSERT INTO test01 VALUES (456, 'good');
INSERT 0 1
```

- Step 4** Run the `\COPY` command to export data from the UTF8 library in Unicode format to the **test01.dat** file.

```
\copy test01 to '/opt/test01.dat' with (ENCODING 'Unicode');
```

- Step 5** Switch to database B and create a table with the same name **test01**.

```
a=> \c b
Password for user dbadmin:
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_128_GCM_SHA256, bits: 128)
You are now connected to database "b" as user "dbadmin".
```

```
b=> CREATE TABLE test01
(
  c_customer_sk      integer,
  c_customer_id      char(5),
  c_first_name       char(6),
  c_last_name        char(8)
)
with (orientation = column,compression=middle)
distribute by hash (c_last_name);
```

**Step 6** Run the `\COPY` command to import the `test01.dat` file to database B.

```
\copy test01 from '/opt/test01.dat' with (ENCODING 'Unicode' ,COMPATIBLE_ILLEGAL_CHARS 'true');
```

#### NOTE

- The error tolerance parameter **COMPATIBLE\_ILLEGAL\_CHARS** specifies that invalid characters are tolerated during data import. Invalid characters are converted and then imported to the database. No error message is displayed. The import is not interrupted.
- The **BINARY** format is not supported. When data of such format is imported, error "cannot specify bulkload compatibility options in BINARY mode" will occur.
- The parameter is valid only for data importing using the **COPY FROM** option.

**Step 7** View data in the `test01` table in database B.

```
b=> select * from test01;
c_customer_sk | c_customer_id | c_first_name | c_last_name
-----+-----+-----+-----
          3769 | hello        | Grace       |
          456  | good        |             |
(2 rows)
```

**Step 8** After the preceding operations are performed, data is imported from database A (UTF8) to database B (GBK).

----End

## 4.7 Can I Import and Export Data to and from OBS Across Regions?

No. No, GaussDB(DWS) does not support OBS data import or export across regions. The GaussDB(DWS) cluster and OBS must be in the same region.

## 4.8 How Do I Import GaussDB(DWS)/Oracle/MySQL/SQL Server Data to GaussDB(DWS) (Whole Database Migration)?

Heterogeneous data can be imported to GaussDB(DWS) using CDM. You can migrate an entire Oracle, MySQL, SQL Server, or GaussDB(DWS) database to a GaussDB(DWS) database. For details, see [Creating an Entire Database Migration Job](#).

You can also store data to OBS and then dump the data to GaussDB(DWS). For details, see [About Parallel Data Import from OBS](#).

## 4.9 Can I Import Data over the Public/External Network Using GDS?

No. The GDS server and GaussDB(DWS) can only communicate with each other on the intranet. Each DN in the GaussDB(DWS) cluster is used to connect to the GDS server in parallel to import a large amount of data. The GDS server and the cluster must be in the same network. If GDS is deployed on an offline server, the firewall needs to be enabled and the cluster needs an EIP. However, one cluster can be bound only to one EIP, and data import with multiple DNs cannot be implemented.

## 4.10 Which Are the Factors That Affect GaussDB(DWS) Import Performance?

The GaussDB(DWS) import performance is affected by the following factors:

1. Cluster specifications: disk I/O, network throughput, memory, and CPU specifications
2. Service planning: type of table fields, compress, and row-store or column-store
3. Data storage: local cluster, OBS
4. Data import mode



# 5 Account, Password, and Permissions

---

## 5.1 How Does GaussDB(DWS) Implement Workload Isolation?

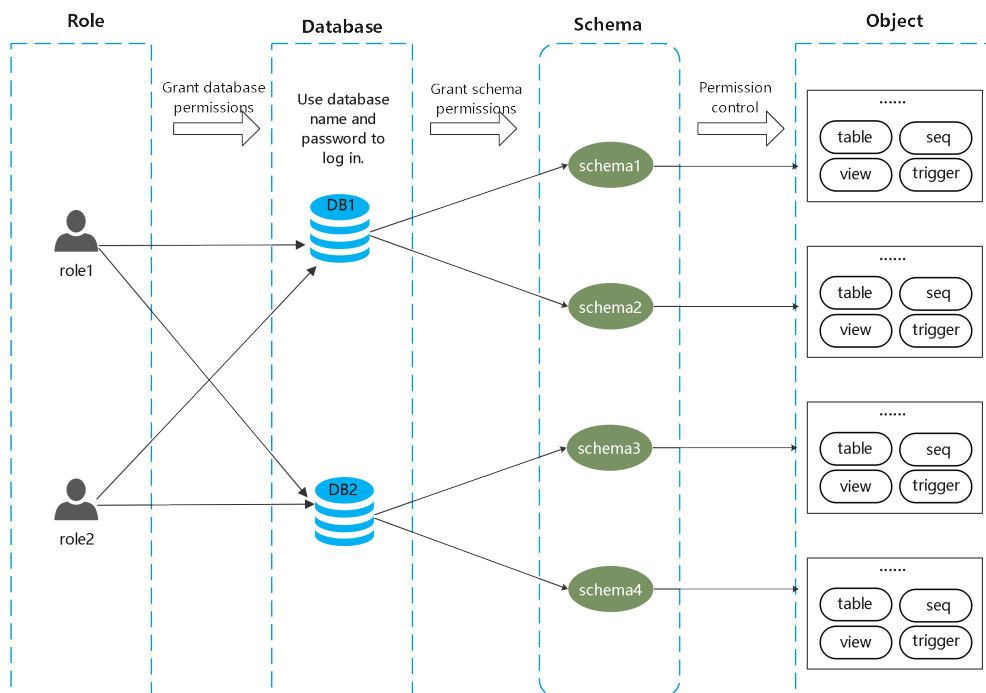
### Workload Isolation

In GaussDB(DWS), you can isolate workloads through database and schema configurations. Their differences are as follows:

- Databases cannot communicate with each other and share very few resources. Their connections and permissions can be isolated.
- Schemas share more resources than databases do. User permissions on schemas and subordinate objects can be flexibly configured 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.

Figure 5-1 Used for permission control.



## DATABASE

A database is a physical collection of database objects. Resources of different databases are completely isolated (except some shared objects). Databases are used to isolate workloads. Objects in different databases cannot access each other. For example, objects in Database B cannot be accessed in Database A. Therefore, when logging in to a cluster, you must connect to the specified database.

## SCHEMA

In a database, database objects are logically divided and isolated based on schemas.

With permission management, you can access and operate objects in different schemas in the same session. Schemas contain objects that applications may access, such as tables, indexes, data in various types, functions, and operators.

Database objects with the same name cannot exist in the same schema, but object names in different schemas can be the same.

```
gaussdb=> CREATE SCHEMA myschema;
CREATE SCHEMA
gaussdb=> CREATE SCHEMA myschema_1;
CREATE SCHEMA

gaussdb=> CREATE TABLE myschema.t1(a int, b int) DISTRIBUTE BY HASH(b);
CREATE TABLE
gaussdb=> CREATE TABLE myschema.t1(a int, b int) DISTRIBUTE BY HASH(b);
ERROR: relation "t1" already exists
gaussdb=> CREATE TABLE myschema_1.t1(a int, b int) DISTRIBUTE BY HASH(b);
CREATE TABLE
```

Schemas logically divide workloads. These workloads are interdependent with the schemas. Therefore, if a schema contains objects, deleting it will cause errors with dependency information displayed.

```
gaussdb=> DROP SCHEMA myschema_1;  
ERROR: cannot drop schema myschema_1 because other objects depend on it  
Detail: table myschema_1.t1 depends on schema myschema_1  
Hint: Use DROP ... CASCADE to drop the dependent objects too.
```

When a schema is deleted, the **CASCADE** option is used to delete the objects that depend on the schema.

```
gaussdb=> DROP SCHEMA myschema_1 CASCADE;  
NOTICE: drop cascades to table myschema_1.t1  
gaussdb=> DROP SCHEMA
```

## USER/ROLE

Users and roles are used to implement permission control on the database server (cluster). They are the owners and executors of cluster workloads and manage all object permissions in clusters. A role is not confined in a specific database. However, when it logs in to the cluster, it must explicitly specify a user name to ensure the transparency of the operation. A user's permissions to a database can be specified through permission management.

A user is the subject of permissions. Permission management is actually the process of deciding whether a user is allowed to perform operations on database objects.

## Permissions Management

Permission management in GaussDB(DWS) falls into three categories:

- System permission

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**.

- 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.

- Object permission

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.

## Schema Isolation Example

Example 1:

By default, the owner of a schema has all permissions on objects in the schema, including the delete permission. The owner of a database has all permissions on objects in the database, including the delete permission. Therefore, you are advised to strictly control the creation of databases and schemas. Create databases and schemas as an administrator and assign related permissions to users.

**Step 1** Assign the permission to create schemas in the **testdb** database to user **user\_1** as user **dbadmin**.

```
testdb=> GRANT CREATE ON DATABASE testdb to user_1;  
GRANT
```

**Step 2** Switch to user **user\_1**.

```
testdb=> SET SESSION AUTHORIZATION user_1 PASSWORD '*****';  
SET
```

Create a schema named **myschema\_2** in the **testdb** database as **user\_1**.

```
testdb=> CREATE SCHEMA myschema_2;  
CREATE SCHEMA
```

**Step 3** Switch to the administrator **dbadmin**.

```
testdb=> RESET SESSION AUTHORIZATION;  
RESET
```

Create **table t1** in schema **myschema\_2** as the administrator **dbadmin**.

```
testdb=> CREATE TABLE myschema_2.t1(a int, b int) DISTRIBUTE BY HASH(b);  
CREATE TABLE
```

**Step 4** Switch to user **user\_1**.

```
testdb=> SET SESSION AUTHORIZATION user_1 PASSWORD '*****';  
SET
```

Delete table **t1** created by administrator **dbadmin** in schema **myschema\_2** as user **user\_1**.

```
testdb=> drop table myschema_2.t1;  
DROP TABLE
```

----End

Example 2:

Due to the logical isolation of schemas, database objects need to be verified at both the schema level and the object level.

**Step 1** Grant the permission on the **myschema.t1** table to **user\_1**.

```
gaussdb=> GRANT SELECT ON TABLE myschema.t1 TO user_1;  
GRANT
```

**Step 2** Switch to user **user\_1**.

```
SET SESSION AUTHORIZATION user_1 PASSWORD '*****';  
SET
```

Query the table **myschema.t1**.

```
gaussdb=> SELECT * FROM myschema.t1;  
ERROR: permission denied for schema myschema  
LINE 1: SELECT * FROM myschema.t1;
```

**Step 3** Switch to the administrator **dbadmin**.

```
gaussdb=> RESET SESSION AUTHORIZATION;  
RESET
```

Grant the permission on the **myschema.t1** table to user **user\_1**.

```
gaussdb=> GRANT USAGE ON SCHEMA myschema TO user_1;  
GRANT
```

**Step 4** Switch to user **user\_1**.

```
gaussdb=> SET SESSION AUTHORIZATION user_1 PASSWORD '*****1';  
SET
```

Query the table **myschema.t1**.

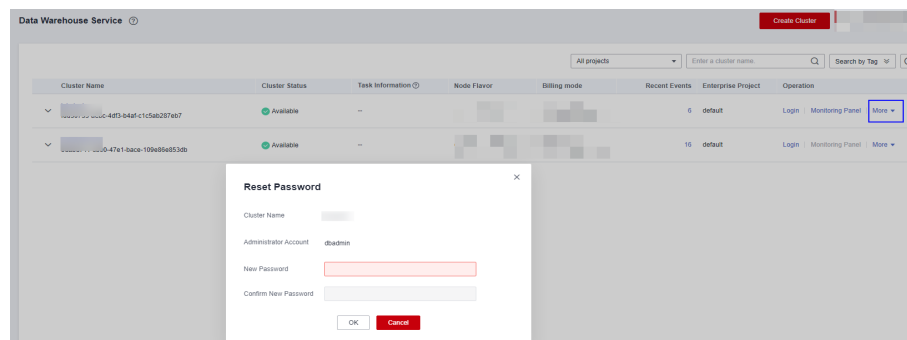
```
gaussdb=> SELECT * FROM myschema.t1;  
a | b  
---+---  
(0 rows)
```

----End

## 5.2 How Do I Change the Password of a Database Account When the Password Expires?

- To change the password of the database administrator **dbadmin**, log in to the console and choose **More** > **Reset Password** in cluster row.

**Figure 5-2** Resetting the password of user dbadmin



For security, the following two parameters manage account passwords. Log in to the console, click the cluster name and switch to the parameter modification page to modify the parameters.

- failed\_login\_attempts**: maximum number of consecutive incorrect password attempts before the account is locked. Run the following statement as user **dbadmin** to unlock the account:  

```
ALTER USER user_name ACCOUNT UNLOCK;
```
- password\_effect\_time**: validity period of the account password, in days. The default value is **90**.

- You can also connect to the database and run the **ALTER USER** command to change the password validity period of a database account (common user and administrator dbadmin).

```
ALTER USER username PASSWORD EXPIRATION 90;
```

## 5.3 How Do I Grant Table Permissions to a User?

This section describes how to grant users the SELECT, INSERT, UPDATE, or full permissions of tables to users.

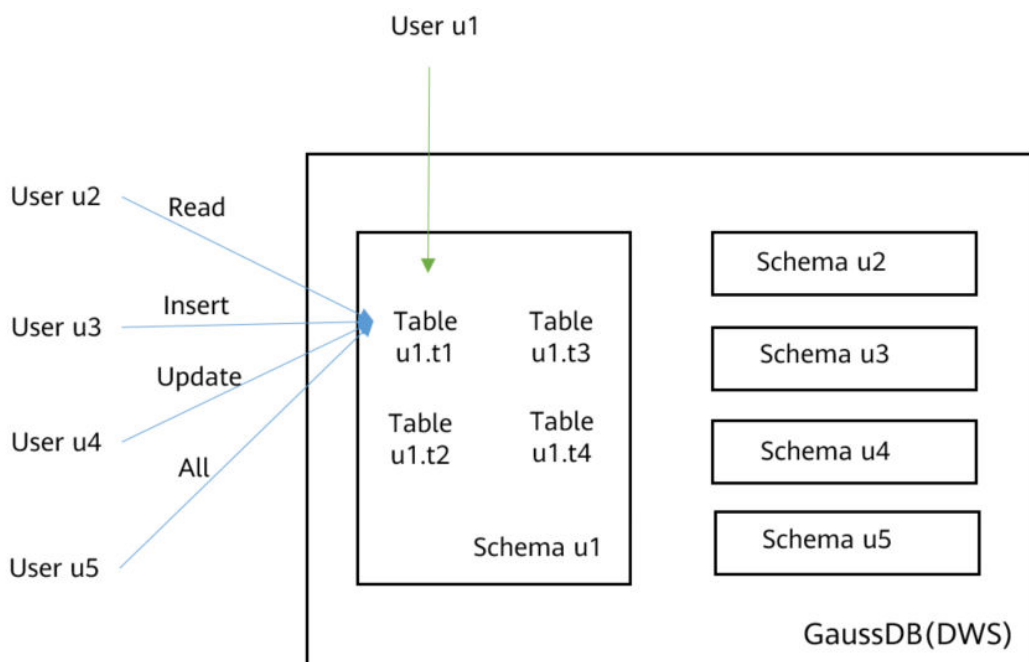
### Syntax

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER | ANALYZE |
ANALYZE } [, ...]
| ALL [ PRIVILEGES ] }
ON { [ TABLE ] table_name [, ...]
| ALL TABLES IN SCHEMA schema_name [, ...] }
TO { [ GROUP ] role_name | PUBLIC } [, ...]
[ WITH GRANT OPTION ];
```

### Scenario

Assume there are users **u1**, **u2**, **u3**, **u4**, and **u5** and five schemas named after these users. Their permission requirements are as follows:

- User **u2** is a read-only user and requires the SELECT permission for the **u1.t1** table.
- User **u3** requires the SELECT permission for the **u1.t1** table.
- User **u3** requires the UPDATE permission for the **u1.t1** table.
- User **u5** requires all permissions of table **u1.t1**.



**Table 5-1** Permissions of the u1.t1 table

Us er	Ty pe	GRANT Statement	Qu ery	Ins ert	Up dat e	Del ete
u1	Ow ner	-	√	√	√	√
u2	Re ad- onl y use r	GRANT SELECT ON u1.t1 TO u2;	√	x	x	x
u3	INS ER T use r	GRANT INSERT ON u1.t1 TO u3;	x	√	x	x
u4	UP DA TE use r	GRANT SELECT,UPDATE ON u1.t1 TO u4; <b>NOTICE</b> The UPDATE permission must be granted together with the SELECT permission, or information leakage may occur.	√	x	√	x
u5	Us ers wit h all per mis sio ns	GRANT ALL PRIVILEGES ON u1.t1 TO u5;	√	√	√	√

## Procedure

Perform the following steps to grant and verify permissions:

- Step 1** Connect to your database as **dbadmin**. Run the following statements to create users **u1** to **u5**. Five schemas will be created and named after the users by default.

```
CREATE USER u1 PASSWORD '{password}';
CREATE USER u2 PASSWORD '{password}';
CREATE USER u3 PASSWORD '{password}';
CREATE USER u4 PASSWORD '{password}';
CREATE USER u5 PASSWORD '{password}';
```

- Step 2** Create table **u1.t1** in schema **u1**.

```
CREATE TABLE u1.t1 (c1 int, c2 int);
```

- Step 3** Insert two records to the table.

```
INSERT INTO u1.t1 VALUES (1,2);  
INSERT INTO u1.t1 VALUES (1,2);
```

**Step 4** Grant schema permissions to users.

```
GRANT USAGE ON SCHEMA u1 TO u2,u3,u4,u5;
```

**Step 5** Grant user **u2** the permission to query the **u1.t1** table.

```
GRANT SELECT ON u1.t1 TO u2;
```

**Step 6** Start a new session and connect to the database as user **u2**. Verify that user **u2** can query the **u1.t1** table but cannot write to or modify the table.

```
SELECT * FROM u1.t1;  
INSERT INTO u1.t1 VALUES (1,20);  
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
```

```
gaussdb=> SELECT * FROM u1.t1;  
 c1 | c2  
----+----  
  1 |  2  
  1 |  2  
(2 rows)  
  
gaussdb=> INSERT INTO u1.t1 VALUES (1,20);  
ERROR: permission denied for relation t1  
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;  
ERROR: permission denied for relation t1
```

**Step 7** In the session started by user **dbadmin**, grant permissions to users **u3**, **u4**, and **u5**.

```
GRANT INSERT ON u1.t1 TO u3; -- Allow u3 to insert data.  
GRANT SELECT,UPDATE ON u1.t1 TO u4; -- Allow u4 to modify the table.  
GRANT ALL PRIVILEGES ON u1.t1 TO u5; -- Allow u5 to query, insert, modify, and delete table data.
```

**Step 8** Start a new session and connect to the database as user **u3**. Verify that user **u3** can query the **u1.t1** table but cannot query or modify the table.

```
SELECT * FROM u1.t1;  
INSERT INTO u1.t1 VALUES (1,20);  
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
```

```
gaussdb=> SELECT * FROM u1.t1;  
ERROR: permission denied for relation t1  
gaussdb=> INSERT INTO u1.t1 VALUES (1,20);  
INSERT 0 1  
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;  
ERROR: permission denied for relation t1
```

**Step 9** Start a new session and connect to the database as user **u4**. Verify that user **u4** can modify and query the **u1.t1** table, but cannot insert data to the table.

```
SELECT * FROM u1.t1;  
INSERT INTO u1.t1 VALUES (1,20);  
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
```



```
gaussdb=> SELECT * FROM u1.t1;
 c1 | c2
----+----
  1 |  2
  1 |  2
  1 | 20
(3 rows)

gaussdb=> INSERT INTO u1.t1 VALUES (1,20);
ERROR: permission denied for relation t1
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
UPDATE 3
```

**Step 10** Start a new session and connect to the database as user **u5**. Verify that user **u5** can query, insert, modify, and delete data in the **u1.t1** table.

```
SELECT * FROM u1.t1;
INSERT INTO u1.t1 VALUES (1,20);
UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
DELETE FROM u1.t1;
```

```
gaussdb=> SELECT * FROM u1.t1;
 c1 | c2
----+----
  1 |  3
  1 |  3
  1 |  3
(3 rows)

gaussdb=> INSERT INTO u1.t1 VALUES (1,20);
INSERT 0 1
gaussdb=> UPDATE u1.t1 SET c2 = 3 WHERE c1 =1;
UPDATE 4
gaussdb=> DELETE FROM u1.t1;
DELETE 4
```

**Step 11** In the session started by user **dbadmin**, execute the `has_table_privilege` function to query user permissions.

```
SELECT * FROM pg_class WHERE relname = 't1';
```

Check the **relacl** column in the command output. *rolename=xxxx/yyyy* indicates that *rolename* has the *xxxx* permission on the table and the permission is obtained from *yyyy*.

The following figure shows the command output.

```
gaussdb=> SELECT * FROM pg_class WHERE relname = 't1';
 relname | relnamespace | reltype | relisatype | relowner | relam | relfilenode | reltablespace | relpages | reltuples | relallvisible | reltoastrelid | reltoastoid | reltoastoid | reldeltarelid | reldeltaoid | relcodescrid | relcodescid | relhasindex |
 relisshared | relpersist | relkind | relnatts | relchecks | relhas2155 | relhaskey | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules | relhasrules |
 relacl |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
 t1      | 2200       | 479682 | 0          | 16393     | 0      | 479680      | 0              | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          |
 r | f |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 t1      | 509400     | 509472 | 0          | 509408    | 0      | 509470      | 7730085       | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          |
 r | f |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 arwdDxtA/u1,u3>=u1,u4=rw/u1 | (orientation=row,compression=no) | d | 8219544 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 r | f |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 t1      | 509400     | 509472 | 0          | 509408    | 0      | 509470      | 7730085       | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          |
 r | f |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 arwdDxtA/u1,u3>=u1,u4=rw/u1 | (orientation=row,compression=no) | d | 8219544 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 r | f |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 t1      | 509400     | 509472 | 0          | 509408    | 0      | 509470      | 7730085       | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          |
 r | f |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
 arwdDxtA/u1,u3>=u1,u4=rw/u1 | (orientation=row,compression=no) | d | 8219544 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
 r | f |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
```

- **u1=arwdDxtA/u1** indicates that **u1** is the owner and has full permissions.
- **u2=r/u1** indicates that **u2** has the read permission.
- **u3=a/u1** indicates that **u3** has the insert permission.
- **u4=rw/u1** indicates that **u4** has the read and update permissions.
- **u5=arwdDxtA/u1** indicates that **u5** has full permissions.

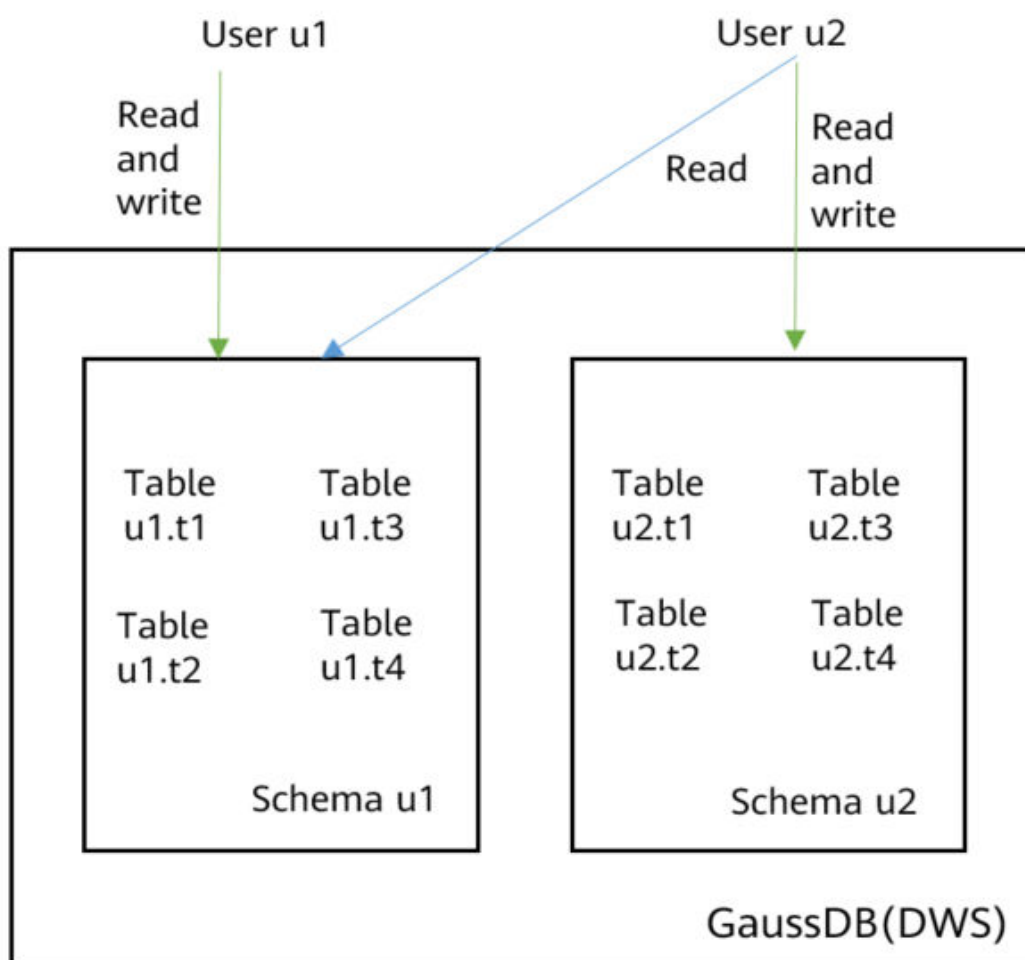
----End

## 5.4 How Do I Grant Schema Permissions to a User?

This section describes how to grant the query permission for a schema as an example. For more information, see [How Do I Grant Table Permissions to a User?](#) :

- Permission for a table in a schema
- Permission for all the tables in a schema
- Permission for tables to be created in the schema

Assume that there are users **u1** and **u2**, and two schemas named after them. User **u2** needs to access tables in schema **u1**.



**Step 1** Connect to your database as **dbadmin**. Run the following statements to create users **u1** and **u2**. Two schemas will be created and named after the users by default.

```
CREATE USER u1 PASSWORD '{password}';
CREATE USER u2 PASSWORD '{password}';
```

**Step 2** Create tables **u1.t1** and **u1.t2** in schema **u1**.

```
CREATE TABLE u1.t1 (c1 int, c2 int);
CREATE TABLE u1.t2 (c1 int, c2 int);
```

**Step 3** Grant the access permission of schema **u1** to user **u2**.

```
GRANT USAGE ON SCHEMA u1 TO u2;
```

**Step 4** Grant user **u2** the permission to query table **u1.t1** in schema **u1**.

```
GRANT SELECT ON u1.t1 TO u2;
```

**Step 5** Start a new session and connect to the database as user **u2** Verify that user **u2** can query the **u1.t1** table but not the **u1.t2** table.

```
SELECT * FROM u1.t1;
SELECT * FROM u1.t2;
```

```
gaussdb=> SELECT * FROM u1.t1;
 c1 | c2
----+----
(0 rows)

gaussdb=> SELECT * FROM u1.t2;
ERROR: permission denied for relation t2
```

**Step 6** In the session started by user **dbadmin**, grant user **u2** the permission to query all the tables in schema **u1**.

```
GRANT SELECT ON ALL TABLES IN SCHEMA u1 TO u2;
```

**Step 7** In the session started by user **u2**, verify that **u2** can query all tables.

```
SELECT * FROM u1.t1;
SELECT * FROM u1.t2;
```

```
gaussdb=> SELECT * FROM u1.t1;SELECT * FROM u1.t2;
 c1 | c2
----+----
(0 rows)

 c1 | c2
----+----
(0 rows)
```

**Step 8** In the session started by user **dbadmin**, create table **u1.t3**.

```
CREATE TABLE u1.t3 (c1 int, c2 int);
```

**Step 9** In the session started by user **u2**, verify that user **u2** does not have the query permission for **u1.t3**. It indicates that user **u2** has the permission to access all the existing tables in schema **u1**, but not the tables to be created in the future.

```
SELECT * FROM u1.t3;
```

```
gaussdb=> SELECT * FROM u1.t3;
ERROR: permission denied for relation t3
```

**Step 10** In the session started by user **dbadmin**, grant user **u2** the permission to query the tables to be created in schema **u1**. Create table **u1.t4**.

 **NOTE**

**ALTER DEFAULT PRIVILEGES** is used to grant permissions on objects to be created.

```
ALTER DEFAULT PRIVILEGES FOR ROLE u1 IN SCHEMA u1 GRANT SELECT ON TABLES TO u2;
CREATE TABLE u1.t4 (c1 int, c2 int);
```

- Step 11** In the session started by user **u2**, verify that user **u2** can access table **u1.t4**, but does not have the permission to access **u1.t3**. To let the user access table **u1.t3**, you can grant permissions by performing [Step 4](#).

```
SELECT * FROM u1.t4;
```

```
gaussdb=> SELECT * FROM u1.t4;
 c1 | c2
----+----
(0 rows)
```

----End

## 5.5 How Do I Create a Database Read-only User?

### Scenario

In service development, database administrators use schemas to classify data. For example, in the financial industry, liability data belong to schema **s1**, and asset data belong to schema **s2**.

Now you have to create a read-only user **user1** in the database. The user can access all tables (including new tables to be created in the future) in schema **s1** for daily reading, but cannot insert, modify, or delete data.

### Principles

DWS provides role-based user management. You need to create a read-only role **role1** and grant the role to **user1**. For details, see [Role-based Access Control](#).

### Procedure

- Step 1** Connect to the DWS database as user **dbadmin**.

- Step 2** Run the following SQL statement to create role **role1**:

```
CREATE ROLE role1 PASSWORD disable;
```

- Step 3** Run the following SQL statement to grant permissions to **role1**:

```
The GRANT usage ON SCHEMA s1 TO role1; -- grants the access permission to schema s1.
GRANT select ON ALL TABLES IN SCHEMA s1 TO role1; -- grants the query permission on all tables in
schema s1.
ALTER DEFAULT PRIVILEGES FOR USER tom IN SCHEMA s1 GRANT select ON TABLES TO role1; -- grants
schema s1 the permission to create tables. tom is the owner of schema s1.
```

- Step 4** Run the following SQL statement to grant the role **role1** to the actual user **user1**:

```
GRANT role1 TO user1;
```

- Step 5** Read all table data in schema **s1** as read-only user **user1**.

----End

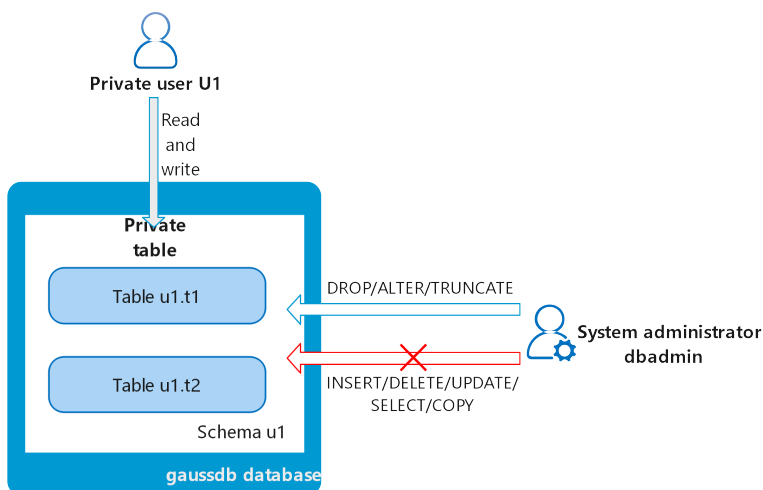
## 5.6 How Do I Create Private Database Users and Tables?

### Scenario

The system administrator **dbadmin** has the permission to access tables created by common users by default. When **Separation of Permissions** is enabled, the administrator **dbadmin** does not have the permission to access tables of common users or perform control operations (DROP, ALTER, and TRUNCATE).

If a private user and a private table (table created by the private user) need to be created, and the private table can be accessed only by the private user and the system administrator **dbadmin** and other common users do not have the permission to access the table (INSERT, DELETE, UPDATE, SELECT, and COPY). However, the system administrator **dbadmin** sometimes need to perform the DROP, ALTER, or TRUNCATE operations without authorization from the private user. In this case, you can create a user (private user) with the INDEPENDENT attribute.

Figure 5-3 Private users



### Principles

This function is implemented by creating a user with the INDEPENDENT attribute.

**INDEPENDENT | NOINDEPENDENT** 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:

- 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 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.

## Procedure

**Step 1** Connect to the DWS database as user **dbadmin**.

**Step 2** Run the following SQL statement to create private user **u1**:

```
CREATE USER u1 WITH INDEPENDENT IDENTIFIED BY 'password';
```

**Step 3** Switch to user **u1**, create the table **test**, and insert data into the table.

```
CREATE TABLE test (id INT, name VARCHAR(20));  
INSERT INTO test VALUES (1, 'joe');  
INSERT INTO test VALUES (2, 'jim');
```

**Step 4** Switch to user **dbadmin** and run the following SQL statement to check whether user **dbadmin** can access the private table **test** created by private user **u1**:

```
SELECT * FROM u1.test;
```

The query result indicates that the user **dbadmin** does not have the access permission. This means the private user and private table are created successfully.

```
gaussdb=> SELECT * FROM u1.test;  
ERROR:  SELECT permission denied to user "dbadmin" for relation "u1.test"
```

**Step 5** Run the **DROP** statement as user **dbadmin** to delete the table **test**.

```
DROP TABLE u1.test;
```

```
gaussdb=> drop table u1.test;  
DROP TABLE
```

----End

## 5.7 How Do I Revoke the CONNECT ON DATABASE Permission from a User?

### Scenario

In a service, the permission of user **u1** to connect to a database needs to be revoked. After the **REVOKE CONNECT ON DATABASE gaussdb FROM u1;** command is executed successfully, user **u1** can still connect to the database. This means the revocation does not take effect.

### Cause Analysis

If you run the **REVOKE CONNECT ON DATABASE gaussdb from u1** command to revoke the permissions of user **u1**, the revocation does not take effect because the

**CONNECT** permission of the database is granted to **PUBLIC**. Therefore, you need to specify **PUBLIC**.

- GaussDB(DWS) provides an implicitly defined group **PUBLIC** that contains all roles. By default, all new users and roles have the permissions of **PUBLIC**. To revoke permissions of **PUBLIC** from a user or role, or re-grant these permissions to them, add the **PUBLIC** keyword in the **REVOKE** or **GRANT** statement.
- GaussDB(DWS) grants the permissions for objects of certain types to **PUBLIC**. By default, permissions on tables, columns, sequences, foreign data sources, foreign servers, schemas, and tablespaces are not granted to **PUBLIC**, but the following permissions are granted to **PUBLIC**;
  - **CONNECT** permission of a database
  - **CREATE TEMP TABLE** permission of a database
  - **EXECUTE** permission of a function
  - **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.

## Example Operations

Run the following command to revoke the permission for user **u1** to access database **gaussdb**:

**Step 1** Connect to the GaussDB(DWS) database **gaussdb**.

```
gsql -d gaussdb -p 8000 -h 192.168.x.xx -U dbadmin -W password -r  
gaussdb=>
```

**Step 2** Create user **u1**.

```
gaussdb=> CREATE USER u1 IDENTIFIED BY 'xxxxxxx';
```

**Step 3** Verify that user **u1** can access GaussDB.

```
gsql -d gaussdb -p 8000 -h 192.168.x.xx -U u1 -W password -r  
gaussdb=>
```

**Step 4** Connect to database **gaussdb** as administrator **dbadmin** and run the **REVOKE** command to revoke the **connect on database** permission of user **public**.

```
gsql -d gaussdb -h 192.168.x.xx -U dbadmin -p 8000 -r  
gaussdb=> REVOKE CONNECT ON DATABASE gaussdb FROM public;  
REVOKE
```

**Step 5** Verify the result. Use **u1** to connect to the database. If the following information is displayed, the **connect on database** permission of user **u1** has been revoked successfully:

```
gsql -d gaussdb -p 8000 -h 192.168.x.xx -U u1 -W password -r  
gsql: FATAL: permission denied for database "gaussdb"  
DETAIL: User does not have CONNECT privilege.
```

----End

## 5.8 How Do I View the Table Permissions of a User?

**Scenario 1:** Run the **information\_schema.table\_privileges** command to **view the table permissions of a user**. Example:

```
SELECT * FROM information_schema.table_privileges WHERE GRANTEE='user_name';
```

```
gaussdb-> SELECT * FROM information_schema.table_privileges WHERE GRANTEE='u2';
grantor | grantee | table_catalog | table_schema | table_name | privilege_type | is_grantable | with_hierarchy
-----|-----|-----|-----|-----|-----|-----|-----
u2      | u2      | gaussdb      | u2           | t2         | INSERT         | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | SELECT         | YES          | YES
u2      | u2      | gaussdb      | u2           | t2         | UPDATE        | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | DELETE        | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | TRUNCATE      | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | REFERENCES    | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | TRIGGER       | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | ANALYZE       | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | VACUUM        | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | ALTER         | YES          | NO
u2      | u2      | gaussdb      | u2           | t2         | DROP          | YES          | NO
u1      | u2      | gaussdb      | u1           | t1         | SELECT        | NO           | YES
(12 rows)
```

**Table 5-2** table\_privileges columns

Column	Data Type	Description
grantor	sql_identifier	Permission grantor
grantee	sql_identifier	Permission grantee
table_catalog	sql_identifier	Database where the table is
table_schema	sql_identifier	Schema where the table is
table_name	sql_identifier	Table name
privilege_type	character_data	Type of the granted permission. The value can be <b>SELECT</b> , <b>INSERT</b> , <b>UPDATE</b> , <b>DELETE</b> , <b>TRUNCATE</b> , <b>REFERENCES</b> , <b>ANALYZE</b> , <b>VACUUM</b> , <b>ALTER</b> , <b>DROP</b> , or <b>TRIGGER</b> .
is_grantable	yes_or_no	Indicates if the permission can be granted to other users. <b>YES</b> indicates that the permission can be granted to other users, and <b>NO</b> indicates that the permission cannot be granted to other users.
with_hierarchy	yes_or_no	Indicates if specific operations are allowed to be inherited at the table level. If the specific operation is <b>SELECT</b> , <b>YES</b> is displayed. Otherwise, <b>NO</b> is displayed.

In the preceding figure, user **u2** has all permissions of table **t2** in schema **u2** and the **SELECT** permission of table **t1** in schema **u1**.

**information\_schema.table\_privileges** can query only the permissions directly granted to the user, the **has\_table\_privilege()** function can query both directly granted permissions and indirect permissions (obtained by GRANT role to user). For example:

```
CREATE TABLE t1 (c1 int);
CREATE USER u1 password '*****';
CREATE USER u2 password '*****';
GRANT dbadmin to u2; //Indirectly grant permissions through roles.
GRANT SELECT on t1 to u1; // Directly grant the permission.

SET ROLE u1 password '*****';
```



```

SELECT * FROM public.t1; // Directly grant the permission to access the table.
c1
----
(0 rows)

SET ROLE u2 password '*****';
SELECT * FROM public.t1; // Indirectly grant the permission to access the table.
c1
----
(0 rows)

RESET role; //Switch back to dbadmin.
SELECT * FROM information_schema.table_privileges WHERE table_name = 't1'; // Can only view direct grants.
 grantor | grantee | table_catalog | table_schema | table_name | privilege_type | is_grantable |
with_hierarchy
-----+-----+-----+-----+-----+-----+-----+-----
dbadmin | u1      | gaussdb      | public      | t1         | SELECT        | NO           | YES
(1 rows)

SELECT has_table_privilege('u2', 'public.t1', 'select'); // Can view both direct and indirect grants.
has_table_privilege
-----
t
(1 row)
    
```

**Scenario 2:** To check whether a user has permissions on a table, perform the following steps:

**Step 1** Query the **pg\_class** system catalog.

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

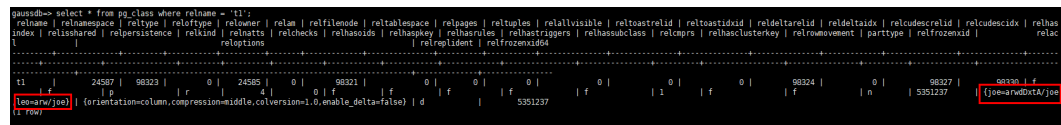
Check the **relacl** column. The command output is shown in the following figure. For details about the permission parameters, see [Table 5-3](#).

- *rolename=xxxx/yyyy*: indicates that *rolename* has the *xxxx* permission on the table and the permission is obtained from *yyyy*.
- *=xxxx/yyyy*: indicates that **public** has the *xxxx* permission on the table and the permission is obtained from *yyyy*.

Take the following figure as an example:

**joe=arwdDxtA**: indicates that user **joe** has all permissions (**ALL PRIVILEGES**).

**leo=arw/joe**: indicates that user **leo** has the read, write, and modify permissions, which are granted by user **joe**.



**Table 5-3** Permissions parameters

Parameter	Description
r	SELECT (read)
w	UPDATE (write)
a	INSERT (insert)

Parameter	Description
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ANALYZE ANALYSE
arwdDxtA	ALL PRIVILEGES (for tables)
*	Actions for preceding permissions

**Step 2** You can also use the **has\_table\_privilege** function to query user permissions on tables.

```
SELECT * FROM has_table_privilege('Username','Table_name','select');
```

For example, query whether user **joe** has the query permission on table **t1**.

```
SELECT * FROM has_table_privilege('joe','t1','select');
```

```
gaussdb=> select * from has_table_privilege('joe','t1','select');
has_table_privilege
-----
 t
(1 row)
```

----End

## 5.9 Who Is User Ruby?

When you run the **SELECT \* FROM pg\_user** statement to view users in the current system, you may see user **Ruby** many times, who has many permissions.

User **Ruby** is an official O&M account. After a GaussDB(DWS) database is created, user **Ruby** is generated by default. There is no security risk in regard to user **Ruby**.

```
gaussdb=> SELECT * FROM pg_user;
username | usesysid | usecreatedb | useuser | usecatupd | use repl | passwd | valbegin | valuntil | respool | parent | spacelimit | useconfig | nodegroup | temppacelimit | spillspa
celimit
-----
 dbadmin | 16384 | f | f | f | f | ***** | | | default_pool | 0 | | | | |
 Ruby | 18 | t | t | t | t | ***** | | | default_pool | 0 | | | | |
 user_1 | 24584 | f | f | f | f | ***** | | | default_pool | 0 | | | | |
 u1 | 24593 | f | f | f | f | ***** | | | default_pool | 0 | | | | |
 u2 | 24597 | f | f | f | f | ***** | | | default_pool | 0 | | | | |
(5 rows)
```

# 6 Database Usage

---

## 6.1 How Do I Change Distribution Columns?

In a data warehouse database, you need to carefully choose distribution columns for large tables, because they can affect your database and query performance. If an improper distribution key is used, data skew may occur after data is imported. As a result, the usage of some disks will be much higher than that of other disks, and the cluster may even become read-only. If the hash distribution policy is used and data skew occurs, the I/O performance of some DN's will be poor, affecting the overall query performance. Proper selection and adjustment of distribution columns are critical to table query performance.

If the hash distribution policy is used, you need to check tables to ensure their data is evenly distributed on each DN. Generally, over 5% difference between the amount of data on different DN's is regarded as data skew. If the difference is over 10%, you have to choose another distribution column.

For tables that are not evenly distributed, adjust their distribution columns to reduce data skew and avoid database performance problems.

### Choosing an Appropriate Distribution Column

The distribution column in a hash table must meet the following requirements, which are ranked by priority in descending order:

- 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 card number column as the distribution key.
- Do not select the column where a constant filter exists.
- Select the join condition as the distribution column, so that join tasks can be pushed down to DN's to execute, reducing the amount of data transferred between the DN's.
- Multiple distribution columns can be selected to evenly distribute data.

## Procedure

Run the **select version();** statement to query the current database version. Required performance varies according to the version.

```
test_lhy=> select version();
                version
-----
 PostgreSQL 9.2.4 (GaussDB 8.1.1 build 7ab61a49) compiled at 2021-06-26 12:05:53 commit 2518 last mr 3356 release
(1 row)
```

- For 8.0.x and earlier versions, rebuild a table.

**Step 1** Use Data Studio or gsql in Linux to access the database.

**Step 2** Create a table.

### NOTE

In the following statements, **table1** is the original table name and **table1\_new** is the new table name. **column1** and **column2** are distribution column names.

```
CREATE TABLE IF NOT EXISTS table1_new
( LIKE table1 INCLUDING ALL EXCLUDING DISTRIBUTION)
DISTRIBUTE BY
HASH (column1, column2);
```

**Step 3** Migrate data to the new table.

```
START TRANSACTION;
LOCK TABLE table1 IN ACCESS EXCLUSIVE MODE;
INSERT INTO table1_new SELECT * FROM table1;
COMMIT;
```

**Step 4** Verify that the table data has been migrated. Delete the original table.

```
SELECT COUNT(*) FROM table1_new;
DROP TABLE table1;
```

**Step 5** Replace the original table.

```
ALTER TABLE table1_new RENAME TO table1;
```

----End

- In 8.1.0 and later versions, you can use the **ALTER TABLE** syntax. For example:

**Step 1** Query the table definition. The command output shows that the distribution column of the table is **c\_last\_name**.

```
select pg_get_tabledef('customer_t1');
```

```
gaussdb=> select pg_get_tabledef ('customer_t1');
                pg_get_tabledef
-----
 SET search_path = public;
 CREATE TABLE customer_t1 (
   c_customer_sk integer,
   c_customer_id character(5),
   c_first_name character(6),
   c_last_name character(8)
 )
 WITH (orientation=column, compression=middle, colversion=2.0, enable_delta=false)
 DISTRIBUTE BY HASH(c_last_name)
 TO GROUP group_version1;
(1 row)
```

**Step 2** Try updating data in the distribution column. An error message will be displayed.

```
update customer_t1 set c_last_name = 'Jimmy' where c_customer_sk = 6885;
```

```
gaussdb=> update customer_t1 set c_last_name = 'Jimmy' where c_customer_sk = 6885;  
ERROR: Distributed key column can't be updated in current version
```

- Step 3** Change the distribution column of the table to a column that cannot be updated, for example, **c\_customer\_sk**.

```
alter table customer_t1 DISTRIBUTE BY hash (c_customer_sk);
```

```
gaussdb=> alter table customer_t1 DISTRIBUTE BY hash (c_customer_sk);  
ALTER TABLE
```

- Step 4** Update the data in the old distribution column.

```
update customer_t1 set c_last_name = 'Jimmy' where c_customer_sk = 6885;
```

```
gaussdb=> update customer_t1 set c_last_name = 'Jimmy' where c_customer_sk = 6885;  
UPDATE 1
```

----End

## 6.2 How Do I View and Set the Database Character Encoding?

### Viewing the Database Character Encoding

Use the **server\_encoding** parameter to check the character set encoding of the current database. For example, the character encoding of database **music** is UTF8.

```
music=> show server_encoding;  
server_encoding  
-----  
UTF8  
(1 row)
```

### Setting the Database Character Encoding

#### NOTE

GaussDB(DWS) does not support the modification of the character encoding format of a created database.

If you need to specify the character encoding format of a database, use **template0** and the **CREATE DATABASE** syntax to create a database. To make your database compatible with most characters, you are advised to use the UTF8 encoding when creating a database.

### CREATE DATABASE syntax

```
CREATE DATABASE database_name  
[ [ WITH ] { [ OWNER [=] user_name ] |  
  [ TEMPLATE [=] template ] |  
  [ ENCODING [=] encoding ] |  
  [ LC_COLLATE [=] lc_collate ] |  
  [ LC_CTYPE [=] lc_ctype ] |  
  [ DBCOMPATIBILITY [=] compatibility_type ] |  
  [ CONNECTION LIMIT [=] connlimit ] } [... ] ];
```

- **TEMPLATE [ = ] template**

Indicates the template name, that is, the name of the template to be used to create the database. GaussDB(DWS) creates a database by copying a database template. GaussDB(DWS) has two initial template databases **template0** and **template1** and a default user database .

Value range: an existing database name. If this is not specified, the system copies **template1** by default. Its value cannot be .

---

**NOTICE**

Currently, database templates cannot contain sequences. If sequences exist in the template library, database creation will fail.

- **ENCODING [ = ] encoding**

Character encoding used by the database. The value can be a character string (for example, **SQL\_ASCII**) or an integer number.

If this parameter is not specified, the encoding of the template database is used by default. The encoding of template databases **template0** and **template1** depends on the OS by default. The character encoding of **template1** cannot be changed. To change the encoding, use **template0** to create a database.

Value range: **GBK**, **UTF8**, and **Latin1**

---

**NOTICE**

The character set encoding of the new database must be compatible with the local settings (**LC\_COLLATE** and **LC\_CTYPE**).

## Examples

Create database **music** using UTF8 (the local encoding type is also UTF8).

```
CREATE DATABASE music ENCODING 'UTF8' template = template0;
```

## 6.3 What Do I Do If Date Type Is Automatically Converted to the Timestamp Type During Table Creation?

When creating a database, you can set the **DBCMPATIBILITY** parameter to the compatible database type. The value of **DBCMPATIBILITY** can be **ORA**, **TD**, and **MySQL**, indicating Oracle, Teradata, and MySQL databases, respectively. If this parameter is not specified during database creation, the default value **ORA** is used. In ORA compatibility mode, the date type is automatically converted to timestamp(0).

To avoid such conversion, set the database to the MySQL compatibility mode, which is the only mode that supports the date type. The compatibility mode of an

existing database cannot be changed. It can only be specified during creation of the database. GaussDB(DWS) supports the MySQL compatibility mode in cluster version 8.1.1 and later. To configure this mode, run the following commands:

```
gaussdb=> CREATE DATABASE mydatabase DBCOMPATIBILITY='mysql';
CREATE DATABASE
gaussdb=> \c mydatabase
Non-SSL connection (SSL connection is recommended when requiring high-security)
You are now connected to database "mydatabase" as user "dbadmin".
mydatabase=> create table t1(c1 int, c2 date);
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
```

If the problem cannot be solved by changing the compatibility, you can try to change the column type. For example, insert data of the date type as trings into a table. Example:

```
gaussdb=> CREATE TABLE mytable (a date,b int);
CREATE TABLE
gaussdb=> INSERT INTO mytable VALUES(date '12-08-2023',01);
INSERT 0 1
gaussdb=> SELECT * FROM mytable;
   a      | b
-----+---
2023-12-08 00:00:00 | 1
(1 row)
gaussdb=> ALTER TABLE mytable MODIFY a VARCHAR(20);
ALTER TABLE
gaussdb=> INSERT INTO mytable VALUES('2023-12-10',02);
INSERT 0 1
gaussdb=> SELECT * FROM mytable;
   a      | b
-----+---
2023-12-08 00:00:00 | 1
2023-12-10          | 2
(2 rows)
```

## 6.4 Do I Need to Run VACUUM FULL and ANALYZE on Common Tables Periodically?

Yes.

For tables that are frequently added, deleted, or modified, you need to periodically perform **VACUUM FULL** and **ANALYZE** to reclaim the disk space occupied by updated or deleted data, preventing performance deterioration caused by data bloat and inaccurate statistics.

- Generally, you are advised to perform **ANALYZE** after a large number of **adding or modification** operations are performed on a table.
- After a table is deleted, you are advised to run **VACUUM** rather than **VACUUM FULL**. However, you can run **VACUUM FULL** in some particular cases, such as when you want to physically narrow a table to decrease the occupied disk space after deleting most rows of the table. For details about the differences between **VACUUM** and **VACUUM FULL**, see [VACUUM and VACUUM FULL](#).

### Syntax

Perform **ANALYZE** on a table.

```
ANALYZE table_name;
```

Perform **ANALYZE** on all tables (non-foreign tables) in the database.

```
ANALYZE;
```

Perform **VACUUM** on a table.

```
VACUUM table_name;
```

Perform **VACUUM FULL** on a table.

```
VACUUM FULL table_name;
```

For details, see [VACUUM](#) and [ANALYZE | ANALYZE](#).

#### NOTE

- If the physical space usage does not decrease after you run the **VACUUM FULL** command, check whether there were other active transactions (started before you delete data transactions and not ended before you run **VACUUM FULL**). If yes, run this command again when the transactions have finished.
- In version 8.1.3 or later, **VACUUM/VACUUM FULL** can be invoked on the management plane. For details, see [Intelligent O&M](#).

## VACUUM and VACUUM FULL

In GaussDB(DWS), the **VACUUM** operation is like a vacuum cleaner used to absorb dust. Here, "dust" means old data. If the data is not cleared in a timely manner, the database space will bloat, causing performance deterioration or even system breakdown.

Purposes of VACUUM:

- Solve space bloat: Clear obsolete tuples and corresponding indexes, which include the tuple (and index) of a committed **DELETE** transaction, the old version (and index) of an **UPDATE** transaction, the inserted tuple (and index) of a rolled back **INSERT** transaction, the new version (and index) of an **UPDATE** transaction, and the tuple (and index) of a **COPY** transaction.
- VACUUM FREEZE: Prevents system breakdown caused by transaction ID wraparound. It converts transaction IDs smaller than OldestXmin to freeze xids, update relfrozenxids in a table, and update relfrozenxids and truncate clogs in a database.
- Update statistics: **VACUUM ANALYZE** updates statistics, enabling the optimizer to select a better way to execute SQL statements.

The VACUUM statement includes **VACUUM** and **VACUUM FULL**. Currently, **VACUUM** can only work on row-store tables. **VACUUM FULL** can be used to release space of column-store tables. For details, see the following table.



**Table 6-1 VACUUM and VACUUM FULL**

Item	VACUUM	VACUUM FULL
Clearing space	If the deleted record is at the end of a table, the space occupied by the deleted record is physically released and returned to the operating system. If the data is not at the end of a table, the space occupied by dead tuples in the table or index is set to be available for reuse.	Despite the position of the deleted data, the space occupied by the data is physically released and returned to the operating system. When data is inserted, a new disk page is allocated.
Lock type	Shared lock. The <b>VACUUM</b> operation can be performed in parallel with other operations.	Exclusive lock. All operations based on the table are suspended during execution.
Physical space	Not released	Released
Transaction ID	Not reclaimed	Reclaimed
Execution overhead	The overhead is low and the operation can be executed periodically.	The overhead is high. You are advised to perform it when the disk page space occupied by the database is close to the threshold and the data operations are few.
Effect	It improves the efficiency of operations on the table.	It greatly improves the efficiency of operations on the table.

## 6.5 Do I Need to Set a Distribution Key After Setting a Primary Key?

No, you only need to set the primary key. By default, the first column of the primary key is selected as the distribution key. If both are set, the primary key must contain the distribution key.

## 6.6 Is GaussDB(DWS) Compatible with PostgreSQL Stored Procedures?

Yes.

GaussDB(DWS) is compatible with PostgreSQL stored procedures. For details, see [Stored Procedures](#).

## 6.7 What Are Partitioned Tables, Partitions, and Partition Keys?

**Partitioned table:** Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored on these smaller physical pieces, namely, partitions, instead of the larger logical partitioned table.

**Partition:** In the GaussDB(DWS) distributed system, data partitioning is to horizontally partition table data within a node based on a specified policy. The table is divided into partitions that do not overlap within a specific range.

**Partition key:** A partition key is an ordered set of one or more table columns. The values in the table partition keys are used to determine the data partition that a row belongs to.

## 6.8 How Can I Export the Table Structure?

You are advised to use the Data Studio graphical client to export table data. You can export data from:

- A specific table
- All tables in a schema
- All tables in a database

For details, see [Exporting Table Data](#).

Alternatively, use `gs_dump` and `gs_dumpall` to export data. You can:

- Export a single database:
  - Database-level export
  - Mode-level export
  - Table-level export
- Export all databases:
  - Database-level export
  - Exporting global objects from each database

For details, see [gs\\_dump](#) and [gs\\_dumpall](#).

## 6.9 How Can I Delete Table Data Efficiently?

Yes. **TRUNCATE** is more efficient than **DELETE** for deleting massive data.

For details, see [TRUNCATE](#).

## Function

**TRUNCATE** quickly removes all rows from a database table.

It has the same effect as the unconditional **DELETE**, but **TRUNCATE** is faster, especially for large tables, because it does not scan tables.

## Functions

- **TRUNCATE TABLE** works like a **DELETE** statement with no **WHERE** clause, that is, emptying a table.
- **TRUNCATE TABLE** uses less system and transaction log resources.
  - **DELETE** deletes a row each time, and records each deletion in the transaction log.
  - **TRUNCATE TABLE** deletes all rows in a table by releasing the data page, and only records each releasing of the data page in the transaction log.
- **TRUNCATE**, **DELETE**, and **DROP** are different in that:
  - **TRUNCATE TABLE** deletes content, releases space, but does not delete definitions.
  - **DELETE TABLE** deletes content, but does not delete definitions or release space.
  - **DROP TABLE** deletes content and definitions, and releases space.

## Examples

```
--Create a table.CREATE TABLE tpcds.reason_t1 AS TABLE tpcds.reason;
--Truncate the table.TRUNCATE TABLE tpcds.reason_t1;
--Drop the table.DROP TABLE tpcds.reason_t1;
--Create a partitioned 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 data.
INSERT INTO tpcds.reason_p SELECT * FROM tpcds.reason;
--Truncate the p_05_before partition.
ALTER TABLE tpcds.reason_p TRUNCATE PARTITION p_05_before;
--Truncate the p_15 partition.
ALTER TABLE tpcds.reason_p TRUNCATE PARTITION for (13);
--Truncate the partitioned table.
TRUNCATE TABLE tpcds.reason_p;
-- Delete the table:
DROP TABLE tpcds.reason_p;
```



hash distribution is selected. The distribution column is the first column whose data type can be used as a distribution column.

```
CREATE TABLE warehouse2
(
  W_WAREHOUSE_SK      INTEGER      ,
  W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
  W_WAREHOUSE_NAME   VARCHAR(20)
);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'w_warehouse_sk' as the distribution
column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

SELECT getdistributekey('warehouse2');
getdistributekey
-----
w_warehouse_sk
(1 row)
```

- Scenario 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.

```
CREATE TABLE warehouse3
(
  W_WAREHOUSE_ID     CHAR(16)      NOT NULL,
  W_WAREHOUSE_NAME   VARCHAR(20)
);
NOTICE: The 'DISTRIBUTE BY' clause is not specified. Using 'w_warehouse_id' as the distribution
column by default.
HINT: Please use 'DISTRIBUTE BY' clause to specify suitable data distribution column.
CREATE TABLE

SELECT getdistributekey('warehouse3');
getdistributekey
-----
w_warehouse_id
(1 row)
```

## 6.12 How Do I Replace the Null Result with 0?

When OUTER JOIN (LEFT JOIN, RIGHT JOIN, and FULL JOIN) is executed, the match failure in the outer join generates a large number of NULL values. You can replace these null values with 0.

You can use the **COALESCE** function to do that. This function returns the first non-null parameter value in the parameter list. For example:

```
SELECT coalesce(NULL,'hello');
coalesce
-----
hello
(1 row)
```

Use left join to join the tables **course1** and **course2**.

```
SELECT * FROM course1;
stu_id | stu_name | cour_name
-----+-----+-----
20110103 | ALLEN   | Math
20110102 | JACK   | Programming Design
20110101 | MAX    | Science
(3 rows)
```

```
SELECT * FROM course2;
cour_id | cour_name | teacher_name
-----+-----+-----
      1002 | Programming Design | Mark
      1001 | Science          | Anne
(2 rows)
```

```
SELECT course1.stu_name,course2.cour_id,course2.cour_name,course2.teacher_name FROM course1 LEFT
JOIN course2 ON course1.cour_name = course2.cour_name ORDER BY 1;
stu_name | cour_id | cour_name | teacher_name
-----+-----+-----+-----
ALLEN    |        |          |
JACK     | 1002 | Programming Design | Mark
MAX      | 1001 | Science          | Anne
(3 rows)
```

Use the **COALESCE** function to replace null values in the query result with 0 or other non-zero values:

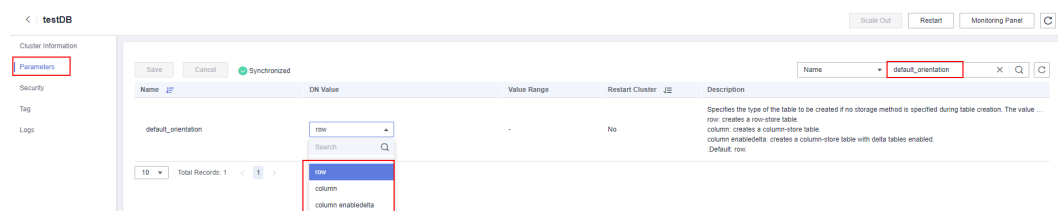
```
SELECT course1.stu_name,
coalesce(course2.cour_id,0) AS cour_id,
coalesce(course2.cour_name,'NA') AS cour_name,
coalesce(course2.teacher_name,'NA') AS teacher_name
FROM course1
LEFT JOIN course2 ON course1.cour_name = course2.cour_name
ORDER BY 1;
stu_name | cour_id | cour_name | teacher_name
-----+-----+-----+-----
ALLEN    | 0 | NA          |
JACK     | 1002 | Programming Design | Mark
MAX      | 1001 | Science          | Anne
(3 rows)
```

## 6.13 How Do I Check Whether a Table Is Row-Stored or Column-Stored?

The storage mode of a table is controlled by the **ORIENTATION** parameter in the table creation statement. **row** indicates row storage, and **column** indicates column storage.

In 8.1.2 and earlier versions, if **ORIENTATION** is not specified, row storage is used by default.

In versions later than 8.1.3, the **default\_orientation** parameter can be used to control the storage mode. If the parameter **ORIENTATION** is not specified during table creation, the table storage mode is based on the value of **default\_orientation**. **row** indicates a row-store table, **column** indicates a column-store table, and **column enabledelta** indicates that a column-store table with delta enabled is created. The GUC parameter can be configured on the GaussDB(DWS) console, as shown in the following figure.



You can use the table definition function **PG\_GET\_TABLEDEF** to check whether the created table is row-store or column-store.

For example, **orientation=column** indicates a column-store table.

Currently, you cannot run the **ALTER TABLE** statement to modify the parameter **ORIENTATION**.

```
SELECT * FROM PG_GET_TABLEDEF('customer_t1');
           pg_get_tabledef
-----
SET search_path = tpchobs;                +
CREATE TABLE customer_t1 (                +
  c_customer_sk integer,                   +
  c_customer_id character(5),              +
  c_first_name character(6),               +
  c_last_name character(8)                 +
)                                           +
WITH (orientation=column, compression=middle, colversion=2.0, enable_delta=false)+
DISTRIBUTE BY HASH(c_last_name)           +
TO GROUP group_version1;
(1 row)
```

## 6.14 How Do I Query the Information About GaussDB(DWS) Column-Store Tables?

The following SQL statements are used to query common information about column-store tables:

Create a column-store partitioned table **my\_table**.

```
CREATE TABLE my_table
(
  product_id INT,
  product_name VARCHAR2(40),
  product_quantity INT
)
WITH (ORIENTATION = COLUMN)
PARTITION BY range(product_quantity)
(
  partition my_table_p1 values less than(600),
  partition my_table_p2 values less than(800),
  partition my_table_p3 values less than(950),
  partition my_table_p4 values less than(1000));

INSERT INTO my_table VALUES(1011, 'tents', 720);
INSERT INTO my_table VALUES(1012, 'hammock', 890);
INSERT INTO my_table VALUES(1013, 'compass', 210);
INSERT INTO my_table VALUES(1014, 'telescope', 490);
INSERT INTO my_table VALUES(1015, 'flashlight', 990);
INSERT INTO my_table VALUES(1016, 'ropes', 890);
```

Run the following command to view the created column-store partitioned table:

```
SELECT * FROM my_table;
product_id | product_name | product_quantity
-----+-----+-----
1013 | compass | 210
1014 | telescope | 490
1011 | tents | 720
1015 | flashlight | 990
1012 | hammock | 890
1016 | ropes | 890
(6 rows)
```

### Querying the Boundary of a Partition

```
SELECT rename, partstrategy, boundaries FROM pg_partition where parentid=(select parentid from
pg_partition where rename='my_table');
```

```
relname | partstrategy | boundaries
-----+-----+-----
my_table | r           |
my_table_p1 | r         | {600}
my_table_p2 | r         | {800}
my_table_p3 | r         | {950}
my_table_p4 | r         | {1000}
(5 rows)
```

## Querying the Number of Columns in a Column-Store Table

```
SELECT count(*) FROM ALL_TAB_COLUMNS where table_name='my_table';
count
-----
3
(1 row)
```

## Querying Data Distribution on DNs

```
SELECT table_skewness('my_table');
table_skewness
-----
("dn_6007_6008 " ,3,50.000%)
("dn_6009_6010 " ,2,33.333%)
("dn_6003_6004 " ,1,16.667%)
("dn_6001_6002 " ,0,0.000%)
("dn_6005_6006 " ,0,0.000%)
("dn_6011_6012 " ,0,0.000%)
(6 rows)
```

## Querying the Names of the Cudesc and Delta Tables in Partition P1 on a DN

```
EXECUTE DIRECT ON (dn_6003_6004) 'select a.relname from pg_class a, pg_partition b where
(a.oid=b.reldeltarelid or a.oid=b.relcudescrid) and b.relname="my_table_p1";
relname
-----
pg_delta_part_60317
pg_cudesc_part_60317
(2 rows)
```

# 6.15 Why Sometimes the GaussDB(DWS) Query Indexes Become Invalid?

Creating indexes for tables can improve database query performance. However, sometimes indexes cannot be used in a query plan. This section describes several common reasons and optimization methods.

### Reason 1: The Returned Result Sets Are Large.

The following uses Seq Scan and Index Scan on a row-store table as an example:

- Seq Scan: searches table records in sequence. All records are retrieved during each scan. This is the simplest and most basic table scanning method, and its cost is high.
- Index Scan: searches the index first, find the target location (pointer) in the index, and then retrieve data on the target page.

Index scan is faster than sequence scan in most cases. However, if the obtained result sets account for a large proportion (more than 70%) of all data, Index Scan needs to scan indexes before reading table data. This makes it slower table scan.



## Reason 2: ANALYZE Is Not Performed In a Timely Manner.

**ANALYZE** is used to update table statistics. If **ANALYZE** is not executed on a table or a large amount of data is added to or deleted from a table after **ANALYZE** is executed, the statistics may be inaccurate, which may cause a query to skip the index.

Optimization method: Run the **ANALYZE** statement on the table to update statistics.

## Reason 3: Filtering Conditions Contains Functions or Implicit Data Type Conversion

If calculation, function, or implicit data type conversion is contained in filter criteria, indexes may fail to be selected.

For example, when a table is created, indexes are created in columns **a**, **b**, and **c**.

```
create table test(a int, b text, c date);
```

- Perform calculation on the indexed columns.

The following command output indicates that both **where a = 101** and **where a = 102 - 1** use the index in column **a**, but **where a + 1 = 102** does not use the index.

```
explain verbose select * from test where a = 101;
QUERY PLAN
```

id	operation	E-rows	E-distinct	E-memory	E-width	E-costs
1	-> Streaming (type: GATHER)	1			44	16.27
2	-> Index Scan using index_a on public.test	1		1MB	44	8.27

Predicate Information (identified by plan id)

```
2 --Index Scan using index_a on public.test
Index Cond: (test.a = 101)
```

Targetlist Information (identified by plan id)

```
1 --Streaming (type: GATHER)
Output: a, b, c
Node/s: dn_6005_6006
2 --Index Scan using index_a on public.test
Output: a, b, c
Distribute Key: a
```

==== Query Summary =====

```
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
```

```
explain verbose select * from test where a = 102 - 1;
QUERY PLAN
```

id	operation	E-rows	E-distinct	E-memory	E-width	E-costs
1	-> Streaming (type: GATHER)	1			44	16.27
2	-> Index Scan using index_a on public.test	1		1MB	44	8.27

Predicate Information (identified by plan id)

```
2 --Index Scan using index_a on public.test
Index Cond: (test.a = 101)
```

```

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: a, b, c
  Node/s: dn_6005_6006
2 --Index Scan using index_a on public.test
  Output: a, b, c
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
explain verbose select * from test where a + 1 = 102;
          QUERY PLAN
-----
id |      operation      | E-rows | E-distinct | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 1 | | | 44 | 22.21
2 | -> Seq Scan on public.test | 1 | | 1MB | 44 | 14.21

Predicate Information (identified by plan id)
-----
2 --Seq Scan on public.test
  Filter: ((test.a + 1) = 102)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: a, b, c
  Node/s: All datanodes
2 --Seq Scan on public.test
  Output: a, b, c
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)

```

Optimization method: Use constants instead of expressions, or put constant calculation on the right of the equal sign (=).

- Use functions on indexed columns.

According to the following execution result, if a function is used on an indexed column, the index fails to be selected.

```

explain verbose select * from test where to_char(c, 'yyyymmdd') =
to_char(CURRENT_DATE,'yyyymmdd');
          QUERY PLAN
-----
id |      operation      | E-rows | E-distinct | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 1 | | | 44 | 22.28
2 | -> Seq Scan on public.test | 1 | | 1MB | 44 | 14.28

          Predicate Information (identified by plan id)
          -----
          2 --Seq Scan on public.test
            Filter: (to_char(test.c, 'yyyymmdd'::text) = to_char(('2022-11-30'::pg_catalog.date)::timestamp
with time zone, 'yyyymmdd'::text))

```

```

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: a, b, c
  Node/s: All datanodes
2 --Seq Scan on public.test
  Output: a, b, c
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
explain verbose select * from test where c = current_date;
          QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----+-----
 1 | -> Streaming (type: GATHER) |      1 |           |          |      44 | 16.27
 2 | -> Index Scan using index_c on public.test |      1 |           |      1MB |      44 | 8.27

Predicate Information (identified by plan id)
-----
2 --Index Scan using index_c on public.test
  Index Cond: (test.c = '2022-11-30'::pg_catalog.date)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: a, b, c
  Node/s: All datanodes
2 --Index Scan using index_c on public.test
  Output: a, b, c
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)

```

Optimization method: Do not use unnecessary functions on indexed columns.

- Implicit conversion of data types.

This scenario is common. For example, the type of column **b** is Text, and the filtering condition is **where b = 2**. During plan generation, the Text type is implicitly converted to the Bigint type, and the actual filtering condition changes to **where b::bigint = 2**. As a result, the index in column **b** becomes invalid.

```

explain verbose select * from test where b = 2;
          QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----+-----
 1 | -> Streaming (type: GATHER) |      1 |           |          |      44 | 22.21
 2 | -> Seq Scan on public.test |      1 |           |      1MB |      44 | 14.21

Predicate Information (identified by plan id)
-----
2 --Seq Scan on public.test
  Filter: ((test.b)::bigint = 2)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)

```

```

Output: a, b, c
Node/s: All datanodes
2 --Seq Scan on public.test
Output: a, b, c
Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
explain verbose select * from test where b = '2';
QUERY PLAN
-----
id | operation | E-rows | E-distinct | E-memory | E-width | E-costs
-----+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 1 | | | 44 | 16.27
2 | -> Index Scan using index_b on public.test | 1 | | 1MB | 44 | 8.27

Predicate Information (identified by plan id)
-----
2 --Index Scan using index_b on public.test
Index Cond: (test.b = '2'::text)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
Output: a, b, c
Node/s: All datanodes
2 --Index Scan using index_b on public.test
Output: a, b, c
Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)

```

Optimization method: Use constants of the same type as the indexed column to avoid implicit type conversion.

### Scenario 4: Hashjoin Is Replaced with Nestloop + Indexscan.

When two tables are joined, the number of rows in the result set filtered by the WHERE condition in one table is small, thus the number of rows in the final result set is also small. In this case, the effect of nestloop+indexscan is better than that of hashjoin. The better execution plan is as follows:

You can see that the Index Cond: (t1.b = t2.b) at layer 5 has pushed the join condition down to the base table scanning.

```

explain verbose select t1.a,t1.b from t1,t2 where t1.b=t2.b and t2.a=4;
id | operation | E-rows | E-distinct | E-memory | E-width | E-costs
-----+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 26 | | | 8 | 17.97
2 | -> Nested Loop (3,5) | 26 | | 1MB | 8 | 11.97
3 | -> Streaming(type: BROADCAST) | 2 | | 2MB | 4 | 2.78
4 | -> Seq Scan on public.t2 | 1 | | 1MB | 4 | 2.62
5 | -> Index Scan using t1_b_idx on public.t1 | 26 | | 1MB | 8 | 9.05
(5 rows)

Predicate Information (identified by plan id)
-----
4 --Seq Scan on public.t2

```

```

Filter: (t2.a = 4)
5 --Index Scan using t1_b_idx on public.t1
  Index Cond: (t1.b = t2.b)
(4 rows)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: t1.a, t1.b
  Node/s: All datanodes
2 --Nested Loop (3,5)
  Output: t1.a, t1.b
3 --Streaming (type: BROADCAST)
  Output: t2.b
  Spawn on: datanode2
  Consumer Nodes: All datanodes
4 --Seq Scan on public.t2
  Output: t2.b
  Distribute Key: t2.a
5 --Index Scan using t1_b_idx on public.t1
  Output: t1.a, t1.b
  Distribute Key: t1.a
(15 rows)

===== Query Summary =====
-----
System available mem: 9262694KB
Query Max mem: 9471590KB
Query estimated mem: 5144KB
(3 rows)

```

If the optimizer does not select such an execution plan, you can optimize it as follows:

```

set enable_index_nestloop = on;
set enable_hashjoin = off;
set enable_seqscan = off;

```

### Reason 5: The Scan Method Is Incorrectly Specified by Hints.

GaussDB(DWS) plan hints can specify three scan method: tablescan, indexscan, and indexonlyscan.

- Table Scan: full table scan, such as Seq Scan of row-store tables and CStore Scan of column-store tables.
- Index Scan: scans indexes and then obtains table records based on the indexes.
- Index-Only Scan: scans indexes, which cover all required results. Compared with the index scan, the index-only scan covers all queried columns. In this way, only indexes are retrieved, and data records do not need to be retrieved.

In Index-Only Scan scenarios, Index Scan specified by a hint will be invalid.

```

explain verbose select /*+ indexscan(test)*/ b from test where b = '1';
WARNING: unused hint: IndexScan(test)
          QUERY PLAN
-----
id | operation | E-rows | E-distinct | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) | 1 | | | 32 | 16.27
2 | -> Index Only Scan using index_b on public.test | 1 | | 1MB | 32 | 8.27

Predicate Information (identified by plan id)
-----
2 --Index Only Scan using index_b on public.test

```

```

Index Cond: (test.b = '1'::text)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: b
  Node/s: All datanodes
2 --Index Only Scan using index_b on public.test
  Output: b
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)
explain verbose select /*+ indexonlyscan(test)*/ b from test where b = '1';
                                QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-memory | E-width | E-costs
---+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) |      1 |           |          |      32 | 16.27
2 | -> Index Only Scan using index_b on public.test |      1 |           |      1MB |      32 | 8.27

Predicate Information (identified by plan id)
-----
2 --Index Only Scan using index_b on public.test
  Index Cond: (test.b = '1'::text)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: b
  Node/s: All datanodes
2 --Index Only Scan using index_b on public.test
  Output: b
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(24 rows)

```

Optimization method: Correctly specify Index scan and Index-Only Scan.

## Reason 6: Incorrect Use of GIN Index in Full-Text Retrieval

To accelerate text search, you can create a GIN index for full-text search.

```
CREATE INDEX idxb ON test using gin(to_tsvector('english',b));
```

When creating the GIN index, you must use the 2-argument version of to\_tsvector. Only when the query also uses the 2-argument version and the arguments are the same as that in the Gin index, the GIN index can be called.

### NOTE

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.

```
explain verbose select * from test where to_tsvector(b) @@ to_tsquery('cat') order by 1;
                                QUERY PLAN
```

```

-----
id |          operation          | E-rows | E-distinct | E-memory | E-width | E-costs
-----+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) |      2 |           |          |         | 44 | 22.23
2 | -> Sort                    |      2 |           | 16MB    | 44 | 14.23
3 | -> Seq Scan on public.test |      1 |           | 1MB     | 44 | 14.21

Predicate Information (identified by plan id)
-----
3 --Seq Scan on public.test
  Filter: (to_tsvector(test.b) @@ "'cat'::tsquery)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: a, b, c
  Merge Sort Key: test.a
  Node/s: All datanodes
2 --Sort
  Output: a, b, c
  Sort Key: test.a
3 --Seq Scan on public.test
  Output: a, b, c
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 1024KB
(29 rows)
explain verbose select * from test where to_tsvector('english',b) @@ to_tsquery('cat') order by 1;
QUERY PLAN
-----
id |          operation          | E-rows | E-distinct | E-memory | E-width | E-costs
-----+-----+-----+-----+-----+-----+-----
1 | -> Streaming (type: GATHER) |      2 |           |          |         | 44 | 20.03
2 | -> Sort                    |      2 |           | 16MB    | 44 | 12.03
3 | -> Bitmap Heap Scan on public.test |      1 |           | 1MB     | 44 | 12.02
4 | -> Bitmap Index Scan      |      1 |           | 1MB     | 0 | 8.00

Predicate Information (identified by plan id)
-----
3 --Bitmap Heap Scan on public.test
  Recheck Cond: (to_tsvector('english'::regconfig, test.b) @@ "'cat'::tsquery)
4 --Bitmap Index Scan
  Index Cond: (to_tsvector('english'::regconfig, test.b) @@ "'cat'::tsquery)

Targetlist Information (identified by plan id)
-----
1 --Streaming (type: GATHER)
  Output: a, b, c
  Merge Sort Key: test.a
  Node/s: All datanodes
2 --Sort
  Output: a, b, c
  Sort Key: test.a
3 --Bitmap Heap Scan on public.test
  Output: a, b, c
  Distribute Key: a

===== Query Summary =====
-----
System available mem: 3358720KB
Query Max mem: 3358720KB
Query estimated mem: 2048KB
(32 rows)

```

Optimization method: Use the 2-argument version of `to_tsvector` for the query and ensure that the argument values are the same as those in the index.

## 6.16 How Do I Use a User-Defined Function to Rewrite the CRC32() Function?

Currently, GaussDB(DWS) does not have a built-in `CRC32()` function. Instead, you can use the user-defined function of GaussDB(DWS) to rewrite the `CRC32()` function.

- `CRC32(expr)`
- Description: Calculates the cyclic redundancy. The input parameter `expr` is a string. If the parameter is NULL, NULL is returned. Otherwise, a 32-bit unsigned value is returned after redundancy calculation.

Example of rewriting the `CRC32()` function using the user-defined function statement of GaussDB(DWS):

```
CREATE OR REPLACE FUNCTION crc32(text_string text) RETURNS bigint AS $$
DECLARE
    val bigint;
    i int;
    j int;
    byte_length int;
    binary_string bytea;
BEGIN
    IF text_string is null THEN
        RETURN null;
    ELSIF text_string = '' THEN
        RETURN 0;
    END IF;

    i = 0;
    val = 4294967295;
    byte_length = bit_length(text_string) / 8;
    binary_string = decode(replace(text_string, E'\\', E'\\\\'), 'escape');
    LOOP
        val = (val # get_byte(binary_string, i))::bigint;
        i = i + 1;
        j = 0;
        LOOP
            val = ((val >> 1) # (3988292384 * (val & 1)))::bigint;
            j = j + 1;
            IF j >= 8 THEN
                EXIT;
            END IF;
        END LOOP;
        IF i >= byte_length THEN
            EXIT;
        END IF;
    END LOOP;
    RETURN (val # 4294967295);
END
$$ IMMUTABLE LANGUAGE plpgsql;
```

Verify the rewriting result.

```
select crc32(null),crc32(''),crc32('1');
crc32 | crc32 |  crc32
-----+-----+-----
| 0 | 2212294583
(1 row)
```

For details about how to use user-defined functions, see [CREATE FUNCTION](#).



## 6.17 What Are the Schemas Starting with pg\_toast\_temp\* or pg\_temp\*?

When you query the schema list, the query result may contain schemas starting with **pg\_temp\*** or **pg\_toast\_temp\***, as shown in the following figure.

```
SELECT * FROM pg_namespace;
```

nspace	nspowner	nspacl	pernspace	usedspace
pg_toast	10	0	-1	0
cstore	10	0	-1	0
gs_logical_cluster	10	0	-1	0
sys	10	0	-1	0
dbas_om	10	0	-1	24576
dbas_lab	10	0	-1	0
pg_catalog	10	0	-1	130088996
public	10	0	-1	822592
information_schema	10	0	-1	352256
utl_file	10	0	-1	0
dbas_output	10	0	-1	0
dbas_random	10	0	-1	0
utl_row	10	0	-1	0
dbas_sql	10	0	-1	0
dbas_lab	10	0	-1	0
scheduler	10	0	-1	8192
u1	24954	0	-1	0
u2	24958	0	-1	0
u3	24962	0	-1	0
u4	24966	0	-1	0
s1	16833	0	-1	0
s2	16833	0	-1	0
pg_temp_om_5003_4_1_201471119284272	10	0	-1	0
pg_toast_temp_cn_5003_4_1_201471119284272	10	0	-1	0
czr_rows	10	0	-1	0

These schemas are created when temporary tables are created. Each session has an independent schema starting with **pg\_temp** to ensure that the temporary tables are visible only to the current session. Therefore, you are not advised to manually delete schemas starting with **pg\_temp** or **pg\_toast\_temp** during routine operations.

Temporary tables are visible only in the current session and are automatically deleted after the session ends. The corresponding schemas are also deleted.

## 6.18 Solutions to Inconsistent GaussDB(DWS) Query Results

In GaussDB(DWS), sometimes a SQL query may get different results. This problem is most likely caused by improper syntax or usage. To avoid this problem, use the syntax correctly. The following are some examples of query results inconsistency along with the solutions.

### Window Function Results Are Incompletely Sorted

#### Scenario:

In the window function **row\_number()**, column **c** of table **t3** is queried after sorting. The two query results are different.

```
select * from t3 order by 1,2,3;
```

```
a | b | c
---+---+---
1 | 2 | 1
1 | 2 | 2
1 | 2 | 3
(3 rows)
```

```
select c,rn from (select c,row_number() over(order by a,b) as rn from t3) where rn = 1;
```

```
c | rn
---+---
1 | 1
(1 row)
```

```
select c,rn from (select c,row_number() over(order by a,b) as rn from t3) where rn = 1;
```

```
c | rn
---+---
3 | 1
(1 row)
```

### Analysis:

As shown above, run **select c,rn from (select c,row\_number() over(order by a,b) as rn from t3) where rn = 1;** twice, the results are different. That is because duplicate values **1** and **2** exist in the sorting columns **a** and **b** of the window function while their values in column **c** are different. As a result, when the first record is obtained based on the sorting result in columns **a** and **b**, the obtained data in column **c** is random, as a result, the result sets are inconsistent.

### Solution:

The values in column **c** need to be added to the sorting.

```
select c,rn from (select c,row_number() over(order by a,b,c) as rn from t3) where rn = 1;
```

```
c | rn
---+---
1 | 1
(1 row)
```

```
select c,rn from (select c,row_number() over(order by a,b,c) as rn from t3) where rn = 1;
```

```
c | rn
---+---
1 | 1
(1 row)
```

## Using Sorting in Subviews/Subqueries

### Scenario

After table **test** and view **v** are created, the query results are inconsistent when sorting is used to query table **test** in a subquery.

```
CREATE TABLE test(a serial ,b int);
INSERT INTO test(b) VALUES(1);
INSERT INTO test(b) SELECT b FROM test;
...
INSERT INTO test(b) SELECT b FROM test;
CREATE VIEW v as SELECT * FROM test ORDER BY a;
```

### Problem SQL:

```
select * from v limit 1;
```

```
a | b
---+---
3 | 1
(1 row)
```

```
select * from (select *from test order by a) limit 10;
```

```
a | b
---+---
14 | 1
(1 row)
```

```
select * from test order by a limit 10;
```

```
a | b
---+---
1 | 1
(1 row)
```

### Analysis:

**ORDER BY** is invalid for subviews and subqueries.

**Solution:**

You are not advised to use **ORDER BY** in subviews and subqueries. To ensure that the results are in order, use **ORDER BY** in the outermost query.

## LIMIT in Subqueries

**Scenario:** When **LIMIT** is used in a subquery, the two query results are inconsistent.

```
select * from (select a from test limit 1 ) order by 1;
a
---
5
(1 row)

select * from (select a from test limit 1 ) order by 1;
a
---
1
(1 row)
```

**Analysis:**

The **LIMIT** in the subquery causes random results to be obtained.

**Solution:**

To ensure the stability of the final query result, do not use **LIMIT** in subqueries.

## Using String\_agg

**Scenario:** When **string\_agg** is used to query the table **employee**, the query results are inconsistent.

```
select * from employee;
empno | ename  | job      | mgr |   hiredate   | sal | comm | deptno
-----+-----+-----+-----+-----+-----+-----+-----
 7654 | MARTIN | SALEMAN | 7698 | 2022-11-08 00:00:00 | 12000 | 1400 |    30
 7566 | JONES  | MANAGER | 7839 | 2022-11-08 00:00:00 | 32000 | 0    |    20
 7499 | ALLEN  | SALEMAN | 7698 | 2022-11-08 00:00:00 | 16000 | 300  |    30
(3 rows)

select count(*) from (select deptno, string_agg(ename, ',') from employee group by deptno) t1, (select
deptno, string_agg(ename, ',') from employee group by deptno) t2 where t1.string_agg = t2.string_agg;
count
-----
      2
(1 row)

select count(*) from (select deptno, string_agg(ename, ',') from employee group by deptno) t1, (select
deptno, string_agg(ename, ',') from employee group by deptno) t2 where t1.string_agg = t2.string_agg;
count
-----
      1
(1 row)
```

**Analysis:**

The **string\_agg** function is used to concatenate data in a group into one row. However, if you use **string\_agg(ename, ',')**, the order of concatenated results needs to be specified. For example, in the preceding statement, **select deptno, string\_agg(ename, ',') from employee group by deptno;**

can output either of the following:

```
30 | ALLEN,MARTIN
```

Or:

```
30 | MARTIN,ALLEN
```

In the preceding scenario, the result of subquery **t1** may be different from that of subquery **t2** when deptno is **30**.

#### Solution:

Add **ORDER BY** to **String\_agg** to ensure that data is concatenated in sequence.

```
select count(*) from (select deptno, string_agg(ename, ',' order by ename desc) from employee group by deptno) t1 ,(select deptno, string_agg(ename, ',' order by ename desc) from employee group by deptno) t2 where t1.string_agg = t2.string_agg;
```

## Database Compatibility Mode

**Scenario:** The query results of empty strings in the database are inconsistent.

database1 (TD-compatible):

```
td=# select " is null;
isnull
-----
f
(1 row)
```

database2 (ORA compatible):

```
ora=# select " is null;
isnull
-----
t
(1 row)
```

#### Analysis:

The empty string query results are different because the syntax of the empty string is different from that of the null string in different database compatibility.

Currently, GaussDB(DWS) supports three types of database compatibility: Oracle, TD, and MySQL. The syntax and behavior vary depending on the compatibility. For details about the compatibility differences, see [Syntax Compatibility Differences Among Oracle, Teradata, and MySQL](#)

Databases in different compatibility modes have different compatibility issues. You can run **select datname, datcompatibility from pg\_database;** to check the database compatibility.

#### Solution:

The problem is solved when the compatibility modes of the databases in the two environments are set to the same. The **DBCMPATIBILITY** attribute of a database does not support **ALTER**. You can only specify the same **DBCMPATIBILITY** attribute when creating a database.

## The configuration item **behavior\_compat\_options** for database compatibility behaviors is configured inconsistently.

**Scenario:** The calculation results of the **add\_months** function are inconsistent.

database1:

```
select add_months('2018-02-28',3) from dual;
add_months
-----
2018-05-28 00:00:00
(1 row)
```

database2:

```
select add_months('2018-02-28',3) from dual;
add_months
-----
2018-05-31 00:00:00
(1 row)
```

### Analysis:

Some behaviors vary according to the database compatibility configuration item **behavior\_compat\_options**. For details about the parameter options, see [behavior\\_compat\\_options](#).

The **end\_month\_calculate** in **behavior\_compat\_options** controls the calculation logic of the **add\_months** function. If this parameter 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**.

### Solution:

The **behavior\_compat\_options** parameter must be configured consistently. This parameter is of the **USERSET** type and can be set at the session level or modified at the cluster level.

## The attributes of the user-defined function are not properly set.

**Scenario:** When the customized function **get\_count()** is invoked, the results are inconsistent.

```
CREATE FUNCTION get_count() returns int
SHIPPABLE
as $$
declare
    result int;
begin
    result = (select count(*) from test); --test table is a hash table.
    return result;
end;
$$
language plpgsql;
```

Call this function.

```
SELECT get_count();
get_count
-----
    2106
(1 row)

SELECT get_count() FROM t_src;
get_count
-----
    1032
(1 row)
```

### Analysis:

This function specifies the **SHIPPABLE** attribute. When a plan is generated, the function pushes it down to DN for execution. The test table defined in the function is a hash table. Therefore, each DN has only part of the data in the table, the result returned by **select count(\*) from test;** is not the result of full data in the test table. The expected result changes after **from** is added.

**Solution:**

Use either of the following methods (the first method is recommended):

1. Change the function to not push down: **ALTER FUNCTION get\_count() not shippable;**
2. Change the table used in the function to a replication table. In this way, the full data of the table is stored on each DN. Even if the plan is pushed down to DN for execution, the result set will be as expected.

## Using the Unlogged Table

**Scenario:**

After an unlogged table is used and the cluster is restarted, the associated query result set is abnormal, and some data is missing in the unlogged table.

**Analysis:**

If **max\_query\_retry\_times** is set to **0** and the keyword **UNLOGGED** is specified during table creation, the created table will be 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. If the cluster restarts unexpectedly (process restart, node fault, or cluster restart), some data in the memory is not flushed to disks in a timely manner, and some data is lost, causing the result set to be abnormal.

**Solution:**

The security of unlogged tables cannot be ensured if the cluster goes faulty. In most cases, unlogged tables are only used as temporary tables. If a cluster is faulty, you need to rebuild the unlogged table or back up the data and import it to the database again to ensure that the data is normal.

## 6.19 Which System Catalogs That the VACUUM FULL Operation Cannot Be Performed on?

**VACUUM FULL** can be performed on all GaussDB(DWS) system catalogs. However, during the process, level 8 locks will be imposed on the system catalogs, and services involving these system catalogs will be blocked.

The suggestions are based on database versions:

### 8.1.3 and Later Versions

- For clusters of version 8.1.3 or later, **AUTO VACUUM** is enabled by default (controlled by the **autovacuum** parameter). After you set the parameter, the system automatically performs **VACUUM FULL** on all system catalogs and row-store tables.
  - If the value of **autovacuum\_max\_workers** is **0**, neither on the system catalogs nor on ordinary tables will **VACUUM FULL** be automatically performed.
  - If **autovacuum** is set to **off**, **VACUUM FULL** will be automatically performed on ordinary tables, but not system catalogs.
- This applies only to row-store tables. To automatically trigger **VACUUM** for column-store tables, you need to configure intelligent scheduling tasks on the management console. For details, see [O&M plan](#).

### 8.1.1 and Earlier Versions

1. Reforming **VACUUM FULL** on the following system catalogs affects all services. Perform this operation in an idle time window or when services are stopped.
  - **pg\_statistic** (Statistics information. You are advised not to clear it because it affects service query performance.)
  - **pg\_attribute**
  - **pgxc\_class**
  - **pg\_type**
  - **pg\_depend**
  - **pg\_class**
  - **pg\_index**
  - **pg\_proc**
  - **pg\_partition**
  - **pg\_object**
  - **pg\_shdepend**
2. The following system catalogs affect resource monitoring and table size query interfaces, but do not affect other services.
  - **gs\_wlm\_user\_resource\_history**
  - **gs\_wlm\_session\_info**
  - **gs\_wlm\_instance\_history**
  - **gs\_respool\_resource\_history**
  - **pg\_relfilenode\_size**
3. Other system catalogs do not occupy space and do not need to be cleared.
4. During routine O&M, you are advised to monitor the sizes of these system catalogs, and collect statistics every week. If the space must be reclaimed, clear the space based on the sizes of the system tables.

The statement is as follows:

```
SELECT c.oid,c.relname, c.relkind, pg_relation_size(c.oid) AS size FROM pg_class c WHERE c.relkind IN ('r') AND c.oid <16385 ORDER BY size DESC;
```

## 6.20 In Which Scenarios Would a Statement Be "idle in transaction"?

When user SQL information is queried in the **PGXC\_STAT\_ACTIVITY** view, the **state** column in the query result sometimes displays **idle in transaction**. **idle in transaction** indicates that the backend is in a transaction, but no statement is being executed. This status indicates that a statement has been executed. Therefore, the value of **query\_id** is 0, but the transaction has not been committed or rolled back. Statements in this state have been executed and do not occupy CPU and I/O resources, but they occupy connection resources such as connections and concurrent connections.

If a statement is in the **idle in transaction** state, rectify the fault by referring to the following common scenarios and solutions:

### Scenario 1: A Transaction Is Started But Not Committed, and the Statement Is in the "idle in transaction" State

**BEGIN/START TRANSACTION** is manually executed to start a transaction. After statements are executed, **COMMIT/ROLLBACK** is not executed. View the **PGXC\_STAT\_ACTIVITY**:

```
SELECT state, query, query_id FROM pgxc_stat_activity;
```

The result shows that the statement is in the **idle in transaction** state.

state	query	query_id
active		0
idle		0
idle		0
active	WLM fetch collect info from data nodes	73464968921613282
active	WLM calculate space info process	0
active	WLM monitor update and verify local info	73464968921613276
active	WLM arbiter sync info by CCN and CNS	0
idle in transaction	select count(1) from t group by a order by 1 desc limit 1;	0
idle		0
active	select state,query,query_id from pgxc_stat_activity;	73464968921613283
active		0
idle		0
idle		0
active	WLM fetch collect info from data nodes	145522562959541153
active	WLM calculate space info process	0
active	WLM monitor update and verify local info	145522562959541123
active	WLM arbiter sync info by CCN and CNS	0
active	SELECT * FROM pg_stat_activity	73464968921613283
idle		0
(19 rows)		

**Solution:** Manually execute **COMMIT/ROLLBACK** on the started transaction.

### Scenario 2: After a DDL Statement in a Stored Procedure Is Executed, Other Nodes of the Stored Procedure Is In the "idle in transaction" State

Create a stored procedure:

```
CREATE OR REPLACE FUNCTION public.test_sleep()
RETURNS void
LANGUAGE plpgsql
AS $$
```



```
BEGIN
  truncate t1;
  truncate t2;
  EXECUTE IMMEDIATE 'select pg_sleep(6)';
  RETURN;
END$$;
```

View the **PGXC\_STAT\_ACTIVITY** view:

```
SELECT coorname,pid,query_id,state,query,username FROM pgxc_stat_activity WHERE username='jack';
```

The result shows that **truncate t2** is in the **idle in transaction** state and **coorname** is **coordinator2**. This indicates that the statement has been executed on **cn2** and the stored procedure is executing the next statement.

coorname	pid	query_id	state	query	username
coordinator1	139767124588288	73464968921614213	active	select test sleep();	jack
coordinator2	140055318353664	0	idle in transaction	truncate t2	jack

(2 rows)

**Solution:** This problem is caused by slow execution of the stored procedure. Wait until the execution of the stored procedure is complete. You can also optimize the statements that are executed slowly in the stored procedure.

### Scenario 3: A Large Number of SAVEPOINT/RELEASE Statements Are in the "idle in transaction" State (Cluster Versions Earlier Than 8.1.0)

View the **PGXC\_STAT\_ACTIVITY** view:

```
SELECT coorname,pid,query_id,state,query,username FROM pgxc_stat_activity WHERE username='jack';
```

The result shows that the **SAVEPOINT/RELEASE** statement is in the **idle in transaction** state.

coorname	pid	query_id	state	query	username
coordinator1	14012787723904	77687093572141691	active	select test sleep1();	jack
coordinator2	139773127153408	0	idle in transaction	release s1	jack
coordinator3	140193352906496	0	idle in transaction	release s1	jack

(3 rows)

**Solution:**

**SAVEPOINT** and **RELEASE** statements are automatically generated by the system when a stored procedure with **EXCEPTION** is executed. In versions later than 8.1.0, **SAVEPOINT** is not delivered to CNs. GaussDB(DWS) stored procedures with **EXCEPTION** are implemented based on subtransactions, the mapping is as follows:

```
begin
  (Savepoint s1)
  DDL/DML
exception
  (Rollback to s1)
  (Release s1)
...
end
```

If there is **EXCEPTION** in a stored procedure when it is started, a subtransaction will be started. If there is an exception during the execution, the current transaction is rolled back and the exception is handled; if there is no exception, the subtransaction is committed.

This problem may occur when there are many such stored procedures and the stored procedures are nested. Similar to scenario 2, you only have to wait after the

entire stored procedure is executed. If there are a large number of **RELEASES** messages, the stored procedure triggers multiple exceptions. In this case, you have to analyze whether the logic of the stored procedure is proper.

## 6.21 How Does GaussDB(DWS) Implement Row-to-Column and Column-to-Row Conversion?

This section describes how to use SQL statements to convert rows to columns and convert columns to rows in GaussDB(DWS).

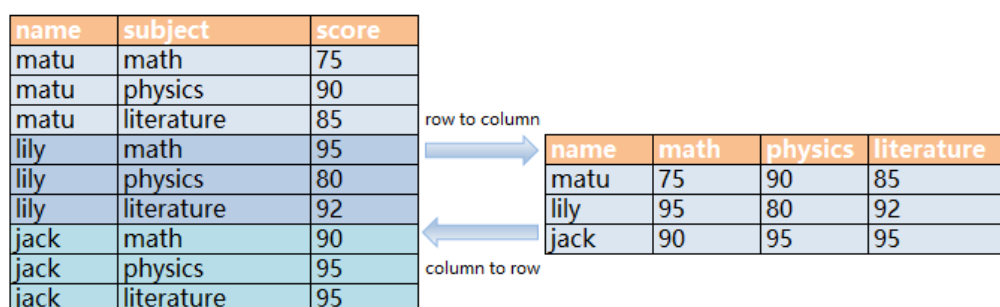
### Scenario

Use a student score table as an example:

Teachers record the score of each subject of each student in a table, but students care only about their own scores. A student needs to use row-to-column conversion to view their scores of all subjects. If the teacher of a subject wants to view the scores of all students of that subject, the teacher needs to use the column-to-row conversion.

The following figure shows the row-to-column and column-to-row conversion.

Figure 6-1 Diagram



- Rows-to-column conversion  
Convert multiple rows of data into one row, or convert one column of data into multiple columns.
- Column-to-row conversion  
Convert a row of data into multiple rows, or convert multiple columns of data into one column.

### Example

- Create a row-store table **students\_info**, and insert data into the table.

```
CREATE TABLE students_info(name varchar(20),subject varchar(100),score bigint) distribute by hash(name);
INSERT INTO students_info VALUES('lily','math',95);
INSERT INTO students_info VALUES('lily','physics',80);
INSERT INTO students_info VALUES('lily','literature',92);
INSERT INTO students_info VALUES('matu','math',75);
INSERT INTO students_info VALUES('matu','physics',90);
```

```
INSERT INTO students_info VALUES('matu','literature',85);
INSERT INTO students_info VALUES('jack','math',90);
INSERT INTO students_info VALUES('jack','physics',95);
INSERT INTO students_info VALUES('jack','literature',95);
```

View information about the **students\_info** table.

```
SELECT * FROM students_info;
name | subject | score
-----+-----+-----
matu | math    | 75
matu | physics | 90
matu | literature | 85
lily | math    | 95
lily | physics | 80
lily | literature | 92
jack | math    | 90
jack | physics | 95
jack | literature | 95
```

- Create a column-store table **students\_info1**, and insert data into the table.  
CREATE TABLE students\_info1(name varchar(20), math bigint, physics bigint, literature bigint) with (orientation = column) distribute by hash(name);  
INSERT INTO students\_info1 VALUES('lily',95,80,92);  
INSERT INTO students\_info1 VALUES('matu',75,90,85);  
INSERT INTO students\_info1 VALUES('jack',90,95,95);

View information about table **students\_info1**.

```
SELECT * FROM students_info1;
name | math | physics | literature
-----+-----+-----+-----
matu | 75 | 90 | 85
lily | 95 | 80 | 92
jack | 90 | 95 | 95
(3 rows)
```

## Static row-to-column conversion

Static row-to-column conversion requires you to manually specify the column names using the given values. If no value is given to a column, the default value 0 is assigned to the column.

```
SELECT name,
sum(case when subject='math' then score else 0 end) as math,
sum(case when subject='physics' then score else 0 end) as physics,
sum(case when subject='literature' then score else 0 end) as literature FROM students_info GROUP BY
name;
name | math | physics | literature
-----+-----+-----+-----
matu | 75 | 90 | 85
lily | 95 | 80 | 92
jack | 90 | 95 | 95
(3 rows)
```

## Dynamic row-to-column conversion

For clusters of 8.1.2 or later, you can use **GROUP\_CONCAT** to generate column-store statements.

```
SELECT group_concat(concat('sum(IF(subject = ', subject, ', score, 0)) AS ', name, '''))FROM students_info;
group_concat
-----
-----
-----
-----
sum(IF(subject = 'literature', score, 0)) AS "jack",sum(IF(subject = 'literature', score, 0)) AS
```

```
"lily",sum(IF(subject = 'literature', score, 0)) AS "matu",sum(IF(subject = 'math', score, 0)) AS "jack",sum(IF
(subject = 'math', score, 0)) AS "lily",sum(IF(subject = 'math', score, 0)) AS "matu",sum(IF(subject =
'physics', score, 0)) AS "jack",sum(IF(subject = 'physics', score, 0)) AS "lily",sum(IF(subject = 'physics
', score, 0)) AS "matu"
(1 row)
```

In 8.1.1 and earlier versions, you can use **LISTAGG** to generate column-store statements.

```
SELECT listagg(concat('sum(case when subject = ', subject, ' then score else 0 end) AS ', subject, ','),',')
within GROUP(ORDER BY 1)FROM (select distinct subject from students_info);
listagg
-----
--
sum(case when subject = 'literature' then score else 0 end) AS "literature",sum(case when subject =
'physics' then score else 0 end) AS "physics",sum(case when subject = 'math' then score else 0 end) AS
"math
"
(1 row)
```

Dynamically rebuild the view:

```
CREATE OR REPLACE FUNCTION build_view()
RETURNS VOID
LANGUAGE plpgsql
AS $$ DECLARE
sql text;
rec record;
BEGIN
sql := 'select LISTAGG(
CONCAT('sum(case when subject = ''', subject, '' then score else 0 end) AS ''', subject, '''' )
,',' ) within group(order by 1) from (select distinct subject from students_info);';
EXECUTE sql INTO rec;
sql := 'drop view if exists get_score';
EXECUTE sql;
sql := 'create view get_score as select name, ' || rec.LISTAGG || ' from students_info group by name';
EXECUTE sql;
END$$;
```

Rebuild the database:

```
CALL build_view();
```

Query view:

```
SELECT * FROM get_score;
name | literature | physics | math
-----+-----+-----+-----
matu |      85 |      90 |      75
lily |      92 |      80 |      95
jack |      95 |      95 |      90
(3 rows)
```

## Column-to-Row Conversion

Use **UNION ALL** to merge subjects (math, physics, and literature) into one column. The following is an example:

```
SELECT * FROM
(
SELECT name, 'math' AS subject, math AS score FROM students_info1
union all
SELECT name, 'physics' AS subject, physics AS score FROM students_info1
union all
SELECT name, 'literature' AS subject, literature AS score FROM students_info1
)
```

```
order by name;
name | subject | score
-----+-----+-----
jack | math     | 90
jack | physics  | 95
jack | literature| 95
lily | math     | 95
lily | physics  | 80
lily | literature| 92
matu | math     | 75
matu | physics  | 90
matu | literature| 85
(9 rows)
```

## 6.22 What Are the Differences Between Unique Constraints and Unique Indexes?

- The concepts of a unique constraint and a unique index are different.  
A unique constraint specifies that the values in a column or a group of columns are all unique. If **DISTRIBUTE BY REPLICATION** is not specified, the column table that contains only unique values must contain distribution columns.  
A unique index is used to ensure the uniqueness of a field value or the value combination of multiple fields. **CREATE UNIQUE INDEX** creates a unique index.
- The functions of a unique constraint and a unique index are different.  
Constraints are used to ensure data integrity, and indexes are used to facilitate query.
- The usages of a unique constraint and a unique index are different.
  - a. Both unique constraints and unique indexes can be used to ensure the uniqueness of column values which can be NULL.
  - b. When a unique constraint is created, a unique index with the same name is automatically created. The index cannot be deleted separately. When the constraint is deleted, the index is automatically deleted. A unique constraint uses a unique index to ensure data uniqueness. GaussDB(DWS) row-store tables support unique constraints, but column-store tables do not.
  - c. A created unique index is independent and can be deleted separately. Currently in GaussDB(DWS), unique indexes can only be created using B-Tree.
  - d. If you want to have both a unique constraint and a unique index on a column, and they can be deleted separately, you can create a unique index and then a unique constraint with the same name.
  - e. If a field in a table is to be used as a foreign key of another table, the field must have a unique constraint (or it is a primary key). If the field has only a unique index, an error is reported.

Example: Create a composite index for two columns, which is not required to be a unique index.

```
CREATE TABLE t (n1 number,n2 number);
CREATE INDEX t_idx ON t(n1,n2);
```

You can use the index **t\_idx** created in the preceding example to create a unique constraint **t\_uk**, which is unique only on column **n1**. A unique constraint is stricter than a unique index.

```
ALTER TABLE t ADD CONSTRAINT t_uk UNIQUE (n1) USING INDEX t_idx;
```

# 7 Database Performance

---

## 7.1 Why Is SQL Execution Slow After Long GaussDB(DWS) Usage?

After a database is used for a period of time, the table data increases as services grow, or the table data is frequently added, deleted, or modified. As a result, bloating tables and inaccurate statistics are incurred, deteriorating database performance.

You are advised to periodically perform **VACUUM FULL** and **ANALYZE** on tables that are frequently added, deleted, or modified. Perform the following operations:

- Step 1** By default, 100 out of 30,000 records of statistics are collected. When a large amount of data is involved, the SQL execution is unstable, which may be caused by a changed execution plan. In this case, the sampling rate needs to be adjusted for statistics. You can run **set default\_statistics\_target** to increase the sampling rate, which helps the optimizer generate the optimal plan.

```
gaussdb=> set default_statistics_target=-2;  
SET
```

- Step 2** Perform **ANALYZE** again. For details, see [ANALYZE | ANALYSE](#).

```
gaussdb=> ANALYZE customer_t1;  
ANALYZE
```

----End

### NOTE

To test whether disk fragments affect database performance, use the following function:

```
select * from pgxc_get_stat_dirty_tables(30,100000);
```

## 7.2 Why Does GaussDB(DWS) Perform Worse Than a Single-Server Database in Extreme Scenarios?

Due to the MPP architecture limitation of GaussDB(DWS), a few PostgreSQL methods and functions cannot be pushed to DN for execution. As a result, performance bottlenecks occur on CNs.

### Explanation:

- An operation can be executed concurrently only when it is logically a concurrent operation. For example, SUM performed on all DN concurrently must centralize the final summarization on one CN. In this case, most of the summarization work has been completed on DN, so the work on the CN is relatively lightweight.
- In some scenarios, the operation must be executed centrally on one node. For example, assigning a globally unique name to a transaction ID is implemented using the system GTM. Therefore, the GTM is also a globally unique component (active/standby). All globally unique tasks are implemented through the GTM in GaussDB(DWS), but software code is optimized to reduce this kind of tasks. Therefore, the GTM does not have many bottlenecks. In some scenarios, GTM-Free and GTM-Lite can be implemented.
- To ensure excellent performance, services need to be slightly modified for adaptation during migration from the application development mode of the traditional single-node database to that of the parallel database, especially for the traditional stored procedure nesting of Oracle.

### Solutions:

- If such a problem occurs, see "Query Performance Optimization" in the [Data Warehouse Service \(DWS\) Developer Guide](#).
- Alternatively, contact technical support to modify and optimize services.

## 7.3 How Can I View SQL Execution Records in a Certain Period When Read and Write Requests Are Blocked?

You can use the top SQL feature to view SQL statements executed in a specified period. SQL statements of the current CN or all CNs can be viewed.

Top SQL allows you to view real-time and historical SQL statements.

- For details about real-time SQL statement query, see section [Real-time TopSQL](#).
- For details about how to query historical SQL statements, see section [Historical TopSQL](#).



## 7.4 What Do I Do If My Cluster Is Unavailable Because of Insufficient Space?

You can use a snapshot to restore your cluster to a new one that has larger storage space, and then delete the old cluster to avoid resource waste. You can learn cluster storage by checking **Available Storage**. To restore your cluster to a new one, see [Restoring a Snapshot to a New Cluster](#)"Restoring a Snapshot to a New Cluster" in *Data Warehouse Service (DWS) User Guide* To delete a cluster, see "Deleting a Cluster" in *Data Warehouse Service (DWS) User Guide*[Deleting a Cluster](#).

### NOTE

This method is only supported by the standard data warehouse.

## 7.5 What is Operator Spilling in GaussDB(DWS)?

During query execution, if the cluster memory is insufficient, the database will choose to store the temporary results to the disk. When the disk storage used by the temporary results exceeds a certain value, users will receive an alarm: "data spilling exceeds the threshold". What does spilling mean?

### Operator Spilling to Disks

Any computing consumes memory space. If too much memory is consumed, the memory for running other jobs will be insufficient, causing unstable job running. Therefore, you need to limit the memory usage of query statements to ensure job running stability.

If a job running requires 500 MB memory but only 300 MB memory is allocated to it, data that is not used temporarily needs to be written to disks, and only data that is being used is retained in the memory. This is called data spilling. It is also called **operator spilling**. Large size of data spilled to disk may cause 100% disk usage. If it happens, the database will turn to read-only and query performance will be greatly affected. Therefore, GaussDB(DWS) imposes a limit on operator spilling. If the operator spilling exceeds the limit, an error is reported and the query exits.

### Which operators can spill?

Operators that can spill to disk include **Hash(VecHashJoin)**, **Agg(VecAgg)**, **Sort(VecSort)**, **Material(VecMaterial)**, **SetOp(VecSetOp)**, and **WindowAgg(VecWindowAgg)**. They can be vectorized or non-vectorized.

### Which parameters can be used to control the operator spilling?

- **work\_mem**: sets spilling threshold. Disk usage exceeding this parameter will cause operator spilling. This parameter takes effect only when the memory is not self-adaptive. (**enable\_dynamic\_workload=off**). It ensures concurrent throughput and the performance of a single query job. Therefore, you need to optimize the parameter based on the output of **Explain Performance**.

- **temp\_file\_limit**: limits the size of files spilled to disks. You are advised to set this parameter based on the site requirements to prevent spilled files from using up the disk space. Spilled files exceeding the value of this parameter will cause an error.

## How Do I Know Whether a Statement Is Spilled to Disks?

- Check the spill files. The spill files are stored in the **base/pgsql\_tmp** directory of the instance directory. The spill files are named *pgsql\_tmp\$queryid\_\$pid*. You can determine which SQL statement is spilled to the disk based on the **queryid**.
- Check the waiting view (**pgxc\_thread\_wait\_status**). If **write file** is displayed in the waiting view, there are temporary results spilled to disks.
- Check the execution plan (**EXPLAIN PERFORMANCE**). Keywords such as **spill**, **written disk**, and **temp file num** indicates that there is an operator spilling.
- Check whether the **spill\_info** column in **Real-time TopSQL** or **Historical TopSQL** contains spill information. If this column is not empty, data has been spilled to disks on DNs. (Prerequisite: The topsql function has been enabled.)

## How Do I Avoid Spilling?

When operators are spilled to disks, operator calculation data is written to disks. Compared with memory access, disk operations are slow, causing performance deterioration and query response time deterioration. Therefore, try to avoid operator spilling during query execution. You are advised to use the following methods:

- Reduce the intermediate result set: If the intermediate result set is too large, you can add filter criteria to reduce the size of the intermediate result set.
- Avoid data skew: If there is a severe data skew, spilling will occur on a DN that has a large amount of data.
- Perform **ANALYZE** in a timely manner: When the statistics are inaccurate, the number of rows may be estimated to be smaller. As a result, the plan is not optimal and data is spilled to disks.
- Perform single-point optimization: Perform optimization on single SQL statements.
- If the memory is not self-adaptive, and the intermediate result set cannot be reduced, increase the value of **work\_mem** as proper.
- If the memory is self-adaptive, increase the available memory of the database as much as possible to reduce the probability of data spilling.

# 7.6 GaussDB(DWS) CPU Resource Management

## Overview of CPU Resource Management

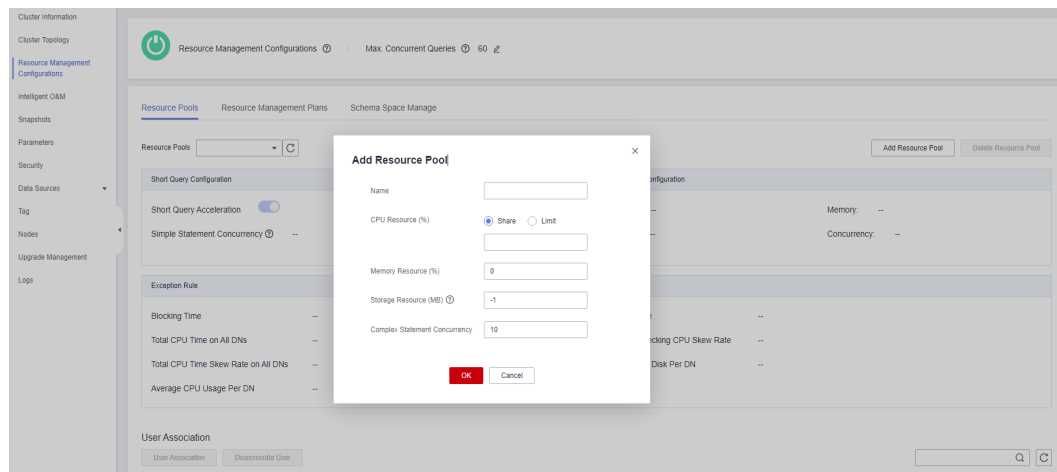
In different service scenarios, system resources (CPU, memory, I/O, and storage resources) of the database are properly allocated to queries to ensure query performance, and service stability.

GaussDB(DWS) provides the resource management function. You can put resources into different resource pools, which are isolated from each other. Then,

you can associate database users with these resource pools. When a user starts a SQL query, the query will be transferred to the resource pool associated with the user. You can specify the number of queries that can be concurrently executed in a resource pool, the upper limit of memory used for a single query, and the memory and CPU resources that can be used by a resource pool. In this way, you can limit and isolate the resources occupied by different workloads.

GaussDB(DWS) uses cgroups to manage and control CPU resources, involving the CPU, `cpuacct`, and `cpuset` subsystems. CPU share is implemented based on the CPU subsystem `cpu.shares`. The advantages of CPU share are as follows: CPU control is not triggered when the OS CPU is not fully occupied. CPU limit is implemented based on `cpuset`, which is a CPU subsystem used to monitor CPU resource usage.

When adding a resource pool on the GaussDB(DWS) management console, you should choose between **Share** and **Limit**.



## CPU Share

**CPU Share:** Percentage of CPU time that can be used by users associated with the current resource pool to execute jobs.

The share has two meanings:

- Share: The CPU is shared by all Cgroups. Idle CPU resources can be used by other Cgroups.
- Quota: When the CPU is fully loaded during peak hours, Cgroups preempt CPU resources based on their quotas.

CPU share is implemented based by `cpu.shares` and takes effect only when the CPU is fully loaded. When the CPU is idle, there is no guarantee that a Cgroup will preempt CPU resources appropriate to its quota. There can still be resource contention when the CPU is idle. Tasks in a Cgroup can use CPU resources without restriction. Although the average CPU usage may not be high, CPU resource contention may still occur at a specific time.

For example, 10 jobs are running on 10 CPUs, and one job is running on each CPU. In this case, any job request for CPU resources will be responded instantly, and there is no contention. If 20 jobs are running on 10 CPUs, the CPU usage may still not be high because the jobs do not always occupy the CPU and may wait for I/O

and network resources. The CPU resources seem idle. However, if 2 or more jobs request one CPU at the same time, CPU resource contention occurs, affecting job performance.

## CPU Limit

**CPU Limit:** specifies the percentage of the maximum number of CPU cores that can be used by a database user in the resource pool.

The limit has two meanings:

- **Dedicated:** The CPU resources are dedicated to a Cgroup. Even the idle part cannot be used by other Cgroups.
- **Quota:** Only the CPU resources in the allocated quota can be used. Idle CPU resources of other Cgroups cannot be preempted.

CPU limit is implemented based on `cpuset.cpu`. You can set a proper quota to implement absolute isolation of CPU resources between Cgroups. In this way, tasks of different Cgroups will not affect each other. However, the absolute CPU isolation will cause idle CPU resources in a Cgroup to be wasted. Therefore, the limit cannot be too large. A larger limit may not bring a better performance.

For example, in one case, 10 jobs are running on 10 CPUs and the average CPU usage is about 5%. In another case, 10 jobs are running on 5 CPUs and the average CPU usage is about 10%. According to the preceding analysis, although the CPU usage is low when 10 jobs run on five CPUs. However, CPU resource contention still exists. Therefore, the performance of running 10 jobs on 10 CPUs is better than that of running 10 jobs on 5 CPUs. However, it is not the more CPUs, the better. If ten jobs run on 20 CPUs, at any time point, at least 10 CPUs are idle. Therefore, theoretically, running 10 jobs on 20 CPUs does not have better performance than running 10 CPUs. For a Cgroup with a concurrency of  $N$ , if the number of allocated CPUs is less than  $N$ , the job performance is better with more CPUs. However, if the number of allocated CPUs is greater than  $N$ , the job performance will not be improved with more CPUs.

## Application Scenarios of CPU Resource Management

The CPU limit and CPU share both have their own advantages and disadvantages. CPU share can fully utilize CPU resources. However, resources of different Cgroups are not completely isolated, which may affect the query performance. CPU limit can implement absolute isolation of CPU resources. However, idle CPU resources will be wasted. Compared with CPU limit, CPU share has higher CPU usage and overall job throughput. Compared with CPU share, CPU limit has complete CPU isolation, which can better meet the requirements of performance-sensitive users.

If CPU contention occurs when multiple types of jobs are running in the database system, you can select different CPU resource control modes based on different scenarios.

- **Scenario 1:** Fully utilize CPU resources. Focus on the overall CPU throughput instead of the performance of a single type of jobs.

Suggestion: You are not advised to isolate CPUs between users. No matter which type of CPU control is implemented, the overall CPU usage is affected.

- Scenario 2: A certain degree of CPU resource contention and performance loss are allowed. When the CPU is idle, the CPU resources are fully utilized. When the CPU is fully loaded, each service type needs to use the CPU proportionally. Suggestion: You can use CPU share to improve the overall CPU usage while implementing CPU isolation and control when the CPUs are fully loaded.
- Scenario 3: Some jobs are sensitive to performance and CPU resource waste is allowed. Suggestion: You can use CPU limit to implement absolute CPU isolation between different types of jobs.

## 7.7 Why the Tasks Executed by an Ordinary User Are Slower Than That Executed by the dbadmin User?

The execution speed of an ordinary user is slower than that of the dbadmin user in the following scenarios:

### Scenario 1: Ordinary users are subject to resource management.

Ordinary users queuing: **waiting in queue/waiting in global queue/waiting in ccn queue**

1. Ordinary users will be **waiting in queue/waiting in global queue** when the number of active statements exceeds the value of **max\_active\_statements**. While administrators do not need to queue. You can increase the value of this parameter or clear some statements to avoid queuing. Change the value of **max\_active\_statements** on the management console.
  - a. Log in to the GaussDB(DWS) management console.
  - b. In the navigation pane on the left, click **Clusters**.
  - c. In the cluster list, find the target cluster and click the cluster name. The **Basic Information** page is displayed.
  - d. Go to the **Parameter Modifications** page of the cluster, search for the **max\_active\_statements** parameter, change its value, and click **Save**.
2. It takes a long time for ordinary users to wait in the ccn queue. When dynamic resource management is enabled (**enable\_dynamic\_workload** is set to **on**), if the concurrency is high and the available memory is small, ordinary users may get into this state when executing statements. Administrators are not controlled. You can stop some statements or increase the memory parameter value. If the memory usage of each DN is not high, you can disable the dynamic resource management parameter **enable\_dynamic\_workload** by setting it to **off**.

### Scenario 2: The OR condition in the execution plan checks the statements executed by common users one by one. This consumes a lot of time.

The **OR** conditions in the execution plans contain permission-related checks. This scenario usually occurs when the system view is used. For example, in the following SQL statement:



By default, GaussDB(DWS) takes the first column of the primary key as the distribution key. If both are set, the primary key must contain the distribution key. Distribution keys determine data distribution among partitions. If distribution keys are well distributed among partitions, query performance can be improved.

If the distribution key is incorrectly selected, data skew may occur after data is imported. The usage of some disks may be much higher than that of other disks, and the cluster may become read-only in some extreme cases. Proper selection of distribution keys is critical to table query performance. In addition, proper distribution keys enable data indexes to be created and maintained more quickly.

## 2. Data Volume Stored in a Single Table

The larger the amount of data stored in a single table, the poorer the query performance. If a table contains a large amount of data, you need to store the data in partitions. To convert an ordinary table to a partitioned table, you need to create a partitioned table and import data to it from the ordinary table. When you design tables, plan whether to use partitioned tables based on service requirements.

To partition a table, comply with the following principles:

- Use fields with obvious ranges for partitioning, for example, date or region.
- The partition name must reflect the data characteristics of the partition. For example, its format can be Keyword+Range characteristics.
- Set the upper limit of a partition to **MAXVALUE** to prevent data overflow.

## 3. Number of Partitions

Tables and indexes can be divided into smaller and easier-to-manage units. This significantly reduces search space and improves access performance.

The number of partitions affects the query performance. If the number of partitions is too small, the query performance may deteriorate.

GaussDB(DWS) supports range partitioning and list partitioning. In range partitioning, records are divided and inserted into multiple partitions of a table. Each partition stores data of a specific range (ranges in different partitions do not overlap). List partitioning is supported only by clusters of 8.1.3 and later versions.

When designing a data warehouse, you need to consider these factors and perform experiments to determine the optimal design scheme.

# 8 Snapshot Backup and Restoration

---

## 8.1 Why Is Creating an Automated Snapshot So Slow?

This happens when the data to be backed up is large. Automated snapshots are incremental backups, and the lower the frequency you set (for example, one week), the longer it takes. Increase backup frequency to speed up the process.

The following table lists the snapshot backup and restoration rates. (The rates are obtained from the lab test environment with local SSDs as the backup media. The rates are for reference only. The actual rate depends on your disk, network, and bandwidth resources.)

- Backup rate: 200 MB/s/DN
- Restoration rate: 125 MB/s/DN

## 8.2 Does a DWS Snapshot Have the Same Function as an EVS Snapshot?

No.

GaussDB(DWS) snapshots are used to restore all the configurations and service data of a cluster. EVS snapshots are used to restore the service data of a data disk or system disk within a specific time period.

### DWS Snapshot

A GaussDB(DWS) snapshot is a full or incremental backup of a GaussDB(DWS) cluster at a specific point in time. It records the current database data and cluster information, including the number of nodes, node specifications, and administrator name. Snapshots can be created manually or automatically.

When a snapshot is used for restoration, GaussDB(DWS) creates a new cluster based on the cluster information recorded in the snapshot and restores data from the snapshot.

For more information about snapshots, see [Managing Snapshots](#).



## EVS snapshot

An EVS snapshot is a complete copy or image of the disk data taken at a specific time point. Snapshot is a major disaster recovery approach, and you can completely restore data of a snapshot to the time when the snapshot was created.

You can create snapshots to rapidly save the disk data at specified time points. In addition, you can use snapshots to create new disks so that the created disks will contain the snapshot data in the beginning.

You can create snapshots to rapidly save the disk data at specified time points to implement data disaster recovery.

- If data loss occurs, you can use a snapshot to completely restore the data to the time point when the snapshot was created.
- You can use snapshots to create new disks so that the created disks will contain the snapshot data.

For more information about snapshots, see [EVS Snapshots](#).

# 9 Billing

---

## 9.1 How Do I Renew the Service?

### About Renewal

Currently, GaussDB(DWS) supports the pay-per-use billing mode and yearly/monthly billing mode.

- In yearly/monthly mode, you pay only once when purchasing the service and no extra fees will be charged during your use of the service. After a yearly/monthly subscription expires, the resources will enter a retention period. If you want to continue using the resources, renew the subscription. For details, see .
- Pay-per-use mode: The system settles the fees by hour. You can use the service as long as your account balance is sufficient. If your account balance is insufficient, it will be in arrears. Top up your account in a timely manner.

### Renewal Modes

#### Pay-per-use

To top up your account, perform the following steps:

- Step 1** Log in to the management console.
- Step 2** Choose **Billing Center** > **Renewal** in the upper right of the page.
- Step 3** In the navigation pane on the left, click **Overview**. Then click **Top Up** so your balance is sufficient to continue services.

----End

#### Yearly/Monthly

Perform the following operations to renew your account:

- Step 1** Log in to the management console.
- Step 2** Click **Service List** on the left and go to the **Data Warehouse Service** page.

**Step 3** On the **Clusters** page, locate the target cluster and click **Renew**.

**Step 4** Complete the renewal as prompted.

----End

## 9.2 Is Refund Supported?

Yes.

### Unsubscribing from a Discount Package

**Step 1** Log in to the management console.

**Step 2** In the upper right corner, click **Billing & Costs**.



Billing & Costs Resources Enterprise Developer Tools ICP License Support Service Tickets

**Step 3** In the navigation tree on the left, choose **Orders > Unsubscriptions**.

**Step 4** Find the resource in the list, click **Unsubscribe from Resources** in the row where the resource is located, and unsubscribe from the resource as prompted.

For details, go to [Unsubscription Rules](#).

----End

## 9.3 How Am I Billed for Scheduled Synchronization of GaussDB(DWS) Data to a PostgreSQL Database?

You can choose either of the following methods to periodically synchronize GaussDB(DWS) data to a PostgreSQL database.

- Use GaussDB(DWS) to export data to OBS, and then import the data from OBS to the PostgreSQL database to implement data synchronization. In this process, fees are generated only when data is stored in OBS. For details about the billing of OBS data storage, see [Object Storage Service Pricing Details](#).

## 9.4 How Can I Try Out GaussDB(DWS) for Free?

Only new users can participate in the free trial campaign. If your account has never created a data warehouse cluster but passed the real-name authentication, you are eligible for a one-month free trial of GaussDB(DWS).

To apply for the free trial, log in to the GaussDB(DWS) management console and click **Apply for free trial**. Free trial packages cannot be used across regions. Select the region based on your requirements.

After subscribing to a free trial package, you can log in to the GaussDB(DWS) management console to create a cluster with the corresponding region, node flavor, and node quantity within the free trial period. The system will not bill you for the created cluster. If you choose to use other node types, you will be charged at the standard pay-per-use rate. For details, see [GaussDB \(DWS\) Pricing Details](#).

When the one-month free trial ends, you can delete the cluster to avoid extra expenses. Alternatively, you can keep the cluster, but you will pay for it at the standard pay-per-use rate.

## 9.5 Why Was I Deducted Fees After My GaussDB(DWS) Free Trial Expired?

As stated on the GaussDB(DWS) activity page, the cluster you created will not be automatically released after the free trial expires. If you do not delete the cluster manually, the system will deduct the fee for the cluster.

To avoid additional fees, log in to the GaussDB(DWS) management console and go to the **Clusters** page to delete your cluster if you do not want to use it after the trial period ends. If the cluster has an EIP, select **Release the EIP bound to the cluster.** on the **Delete Cluster** dialog box. If you do not release the EIP, it will be billed as per VPC EIP pricing rule.

## 9.6 Why Can't I See a Cluster After I Subscribe to a Free GaussDB(DWS) Trial?

The system does not automatically create a cluster after you subscribe to a free trial. Log in to the GaussDB(DWS) management console to manually create one.

## 9.7 How Can I Stop GaussDB(DWS) Billing?

The following describes how to stop billing in pay-per-use and yearly/monthly.

- **Pay-per-use**

If you no longer use a GaussDB(DWS) cluster that is in pay-per-use mode, delete it and its resources to stop the billing. To do so, log in to the GaussDB(DWS) management console and go to the **Clusters** page. The following table lists the billable items of GaussDB(DWS).

**Table 9-1** GaussDB(DWS) billable items

Billing Item	Description
Data warehouse node	The billing will be stopped after the cluster is deleted. For details about how to delete a cluster, see <a href="#">Deleting a Cluster</a> .

Billing Item	Description
Snapshot storage space	When a cluster is deleted, its automated snapshots are deleted, but its manual snapshots are retained. After you delete the manual snapshots, the billing is stopped. To delete the manual snapshots, log in to the GaussDB(DWS) management console and go to the <b>Clusters</b> page. If the manual snapshots are still retained after the cluster is deleted and the total snapshot size exceeds the free space, the excess part will be billed based on the OBS billing rate.
(Optional) EIP and bandwidth	If a cluster has an EIP, select <b>Release the EIP bound to the cluster</b> . when deleting the cluster. The billing will be stopped after the EIP is released. If you do not release the EIP, it will be billed as per VPC EIP pricing rules.
(Optional) DEK	If you have enabled the <b>Encrypt DataStore</b> function for a cluster and purchased a key on DEW, the key is not deleted after you delete the cluster. Manually unsubscribe from and delete the key to stop billing. To do this, log in to the DEW management console and choose <b>Data Encryption Workshop &gt; Key Pair Service</b> .

- Yearly/Monthly package

GaussDB(DWS) clusters billed in yearly/monthly mode support unconditional unsubscription within five days from your purchase. For details, see . If you do not unsubscribe from the package, the system does not refund the fees.

## 9.8 Does Pay-per-Use Billing Stop When My Cluster Stops?

No. The system settles the fees by hour for the pay-per-use mode. You can use the service as long as your account balance is sufficient. To reduce costs:

- Delete the purchased clusters if you will not be needing them, and create new ones when needed.
- Switch to the yearly/monthly mode that allows you to use the service within the specific period of time without additional fees.

## 9.9 Why Is the Purchase Button Unavailable When I Create a Cluster?

The possible causes for unavailable purchase button during cluster creation are as follows:

- The flavor to be purchased could be sold out, or the region does not have such a flavor.

**Solution:** Before purchasing, check whether the flavor is available in the region, or purchase a package after the cluster is created. After the package, the package is automatically associated with the cluster.

- The account could be in arrears or restricted, so new resources cannot be created.

**Solution:** If your balance is insufficient, top up the account to write off the overdue amount.

## 9.10 How Do I Unfreeze a Cluster?

### Cause Analysis

If your account balance is insufficient and fee deduction fails, the retention period starts. During the retention period, the service resources will be frozen and cannot be used, but resources and data are reserved.

### Handling Procedure

To unfreeze a cluster, you need to top up your account to ensure that the account balance is not **0**. For details, see [How Do I Renew the Service?](#) After a cluster is unfrozen, the cluster status changes to **Available**.

## 9.11 Can I Freeze or Shut Down a GaussDB(DWS) Cluster to Stop Billing?

No. If you do not need a cluster, delete it and its resources.